AFRL-RY-WP-TR-2012-0038

BOOTSTRAPPED LEARNING ANALYSIS AND CURRICULUM DEVELOPMENT ENVIRONMENT (BLADE)

Howard Reubenstein, Dan Hunter, and Kathy Ryall

BAE Systems, Inc.

FEBRUARY 2012
Final Report

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

NOTICE AND SIGNATURE PAGE

*//Signature//
DR. PHILIP D. MUMFORD, Project Engineer
Distributed Collaborative Sensor System
Technology Branch

//Signature//
JUAN M. CARBONELL, Chief
Distributed Collaborative Sensor System
Technology Branch

//Signature//
TODD A. KASTLE, Chief
Integrated Electronic and Net-Centric Warfare Division
Sensors Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YY)* February 2012	**2. REPORT TYPE** Final	**3. DATES COVERED *(From - To)*** 31 July 2007 – 31 October 2011

4. TITLE AND SUBTITLE
BOOTSTRAPPED LEARNING ANALYSIS AND CURRICULUM DEVELOPMENT ENVIRONMENT (BLADE)

5a. CONTRACT NUMBER FA8650-07-C-7722
5b. GRANT NUMBER
5c. PROGRAM ELEMENT NUMBER 62304E

6. AUTHOR(S)
Howard Reubenstein, Dan Hunter, and Kathy Ryall

5d. PROJECT NUMBER 2000
5e. TASK NUMBER SC
5f. WORK UNIT NUMBER BOLESC01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
BAE Systems, Inc.
6 New England Executive Park
Burlington, MA 01803

8. PERFORMING ORGANIZATION REPORT NUMBER TR-2764

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory
Sensors Directorate
Wright-Patterson Air Force Base, OH 45433-7320
Air Force Materiel Command
United States Air Force

Defense Advanced Research Projects Agency
Information Innovation Office (DARPA/I2O)
3701 N. Fairfax Drive
Arlington, VA 22203-1714

10. SPONSORING/MONITORING AGENCY ACRONYM(S) AFRL/RYWC

11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S) AFRL-RY-WP-TR-2012-0038

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES
DARPA PAO Case Number: 18991, cleared 06 April 2012. ©2012 BAE Systems, Inc. This work was funded in whole or in part by Department of the Air Force Contract FA8650-07-C-7722. The U.S. Government has for itself and others acting on its behalf a paid-up, nonexclusive, irrevocable worldwide license to use, modify, reproduce, release, perform, display, or disclose the work by or on behalf of the U. S. Government. Report contains color.

14. ABSTRACT
DARPA's Bootstrapped Learning (BL) program was a research effort to build and evaluate an electronic learner (e- student) that can be instructed by a human in the style of human-mentored learning. The BAE Systems' team was responsible for evaluation, curriculum construction, and instructional mechanisms of the BL program. An independent Learning Team was responsible for the machine learning (ML) algorithms. This report presents BAE Systems' results in developing instructional materials to test the e-student, evaluating the e-student's learning results against control human subjects, and developing instructional techniques and mechanisms to teach the e-student. Overall program results demonstrate that natural instruction is a powerful and concise alternative to typical instructional input provided to current ML systems; they also point to the feasibility of constructing individual learning methods that can take advantage of multiple types of instructional input. In the future, we recommend focusing on learning and performance in a specific domain (e.g., ISR analysis) as it would (1) provide the opportunity to overcome the limitations discussed in this report, (2) stretch the bootstrapping approach to an extended training regimen (extended over both time and conceptual coverage), and (3) provide the opportunity to assess success in a concrete domain with specific success criteria.

15. SUBJECT TERMS
Bootstrapped Learning, machine learning, instructable computing, learning performance evaluation

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT:**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** (Monitor)
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	SAR	56	Dr. Philip D. Mumford **19b. TELEPHONE NUMBER** *(Include Area Code)* N/A

Table of Contents

List of Figures

List of Tables

Acknowledgements

The following institutions joined BAE Systems as part of our BLADE team and we would like to acknowledge their contributions to the work described herein:

Institution	*Principal*
Cycorp, Inc.	Michael Witbrock & Jon Curtis
University of Texas at Austin	Dewayne Perry
Stottler Henke Associates, Inc.	Dan Fu & Jeremy Ludwig
Soar Technology, Inc.	Mike van Lent
SRI International	Michael Freed
Teknowledge Corporation	Robert Balzer
Sarnoff Corporation	Hui Cheng & Chumki Basu

1.0 Summary

The DARPA Bootstrapped Learning (BL) program [1] was a research effort to build and evaluate an electronic learner who can be instructed by a human instructor in the style of human mentored learning (or, to support automated testing, by a machine instructor using natural instruction methods). The motivating goal of this effort was to support modification of systems via field instructability (versus programmed revision by the developers). The functionality of computer systems and devices as programmed by their developers often is rendered out of date, incomplete or obsolete particularly in the case of military systems fielded to be applied in constantly evolving situations against adversaries who adapt to our tactics. We denote our approach to field instructability as *instructable computing* because the learner is a computational entity acquiring executable knowledge by being taught/instructed rather than being programmed. The goal is to learn using the "same" instruction methods used between humans (as opposed, for example, to traditional machine learning techniques which are based on processing hundreds, if not thousands or more, annotated examples as supervisory input). Human-mentored learning is primarily based upon communication from an instructor, who has knowledge of the target of learning, to a student that does not have such knowledge.

The BAE Systems team was responsible for curriculum construction, instructional mechanisms, and advising the government on the evaluation of the BL program. The co-contractor team led by SRI International was responsible for the machine learning algorithms. The objective of this report is to present BAE Systems' results in developing instructional materials to test the SRI International learner (MABLE) [2], evaluating the results of learning of the electronic student and of control human subjects, and developing instructional techniques and mechanisms used to teach the electronic learner. The report covers a broad range of efforts, including the following four topics which are central to the BL approach:

- **Natural Instruction Methods (NIMs) –** the BL program uses supervisory input that goes well beyond the standard annotated corpora of examples. NIMs are defined protocols that guide the content of information exchange between an instructor and the electronic student. They govern the type of information that can be presented, the control vocabulary for the instructional interaction (though not the information content – this remains flexible), and the order of interaction. A powerful example of a NIM protocol is the relevance annotation which allows specifying attributes or predicates relevant to learning a concept. Relevance declarations allow a learner to focus its efforts and support learning-by-example (and specifically from a very concise set of pedagogically selected examples).
- **Curriculum –** curriculum definitions support automated testing of the learning system. A curriculum is a hierarchical structure for teaching concepts that build upon each other (i.e., bootstrapping). Individual concepts are defined in the curriculum via a set of parameterized instructional utterances that can be used to teach the concept. Curriculum artifacts contained enough detail to enable the automated teacher to effectively instruct the learning system. While the ultimate goal of the project is to support human instruction, the curriculum we developed serve as a corpus of training and testing examples that can be used to support future efforts to develop alternate bootstrapped learning algorithms.

- **Domains** – the BL learning approach is domain independent. Nevertheless, the program needed to choose specific domains and tasks to drive the instructional interaction. We developed a number of domains including: Blocks World, Unmanned Aerial Vehicle (UAV) control, International Space Station (ISS) subsystem control diagnosis, military ground movement planning (Armored Task Force (ATF)), and satellite ground station control diagnosis and repair. Selection of these domains proved critical to developing domain independent learning techniques as they require solving a breadth of problems (as opposed to focusing on a single domain – which would have resulted in teaching, and learning to the test). Proof of the domain independence of the learning techniques was provided by successful performance on the hidden domain (satellite ground station control).
  Development of multiple domains also revealed some fundamental modeling differences for these domains in terms of fundamental features such as: synchrony of actions, percept modeling of domain state, and causal proximity of actions.
- **Situated instructional interface** – as the program proceeded, more emphasis was placed on instruction by human agents (as opposed to the automated teacher). A prototype instructional interface was developed as part of the BL Intelligence, Surveillance, and Reconnaissance (BL-ISR) application domain. Initial efforts at a disembodied conversational interface proved much less successful than the instructional interface situated in the interface of the BL-ISR domain simulator. Situated instruction allows an analyst to perform instruction more effectively as part of the overall task being performed.

This report also describes an ancillary effort executed as part of the program. We present BL-ISR, an application of BAE Systems' instructable computing approach with potential to improve the productivity of ISR analysts.

Finally, we report on a pair of seedling efforts (Probabilistic Relational Ontological Web Language (PROWL) and Framework Adoption) that explored approaches to system longevity, complementing the instructable computing approach taken in the BL program.

The overall results of the BL program demonstrate that natural instructional input is a powerful and concise alternative to typical instructional input provided to current machine learning (ML) systems. Natural instruction provides a variety of methods of teaching including: by-example, by-telling, and by-feedback. Instructional input was provided via curriculum definition, via
command line interface, via situated interface, and even via natural language input. Each of these instructional techniques was validated as capable of providing the information necessary for concept acquisition. However, since the form of instructional input is so much more varied than, e.g., simple positive and negative training data, it does require definition of protocols (natural instructional methods) for the exchange of instructional content, along with interface mechanisms to facilitate provisioning of this information from a context that includes relevant objects and relations. The results also point to the feasibility of constructing individual learning methods that can take advantage of certain types of instructional input.

The greatest remaining challenge, however, is to develop effective methods for individual learners to integrate their partial results to provide a unified result of learning. If the learning task is thought of as literally learning a program to accomplish a particular task or recognize a certain concept (which in fact is what the BL learners produced as their result of learning), then integration of learning results amounts to the different learning modules authoring different partsof the resultant program and combining the individual pieces into a single whole. Integration of learning results remains an unsolved challenge.

2.0 Introduction

The BL program was a Defense Advanced Research Projects Agency (DARPA) initiative aimed at advancing the state of the art in instructable computing. Its primary objective was to build an electronic student (e-student) that could be taught by a human instructor in the same ways that humans instruct one another. As BL would provide natural ways for a human to impart knowledge to a software learner, it would reduce or even remove the need for programming expertise; human instruction of e-students will make it possible to delegate tasks to computers that cannot be easily delegated today and will enable users to rapidly modify deployed systems.

BL differs from other kinds of ML in several ways. Current ML is primarily a modeling tool; it is used to build models when we know something, but not everything, about some target problem. Current ML is a process of discovery, and requires induction over large datasets to induce its models. There is no guarantee that target knowledge will be discovered. BL enables users to impart the target knowledge in a more direct fashion; moreover, because it involves "natural" ways to impart knowledge, it does not require programming expertise. BL supports conceptual bootstrapping; it leads to meaningful intermediary levels of learned concepts. E-students learn laddered-curricula in which lessons build on previous lessons, whereas in current ML, learning is generally from unstructured data. Like its human counterpart, an e-student assumes all necessary knowledge is possessed by the instructor, and its goal is to learn using the "same" instruction methods used between humans.

Two teams performed in parallel to explore this new learning paradigm. Roughly divided, the Learning Team, led by SRI International, was charged with developing the e-student, incorporating several learning strategies [2,3,4]; the Curriculum Team, led by BAE Systems, developed the curricula that the e-student was expected to learn, an electronic teacher to execute the curricula (i.e., automatically instruct), and a framework that supports interactions between the two electronic agents. BAE Systems also conducted experiments to evaluate various aspects of the program, including an evaluation of the e-student for both hidden and known domains. The two teams worked together to define the formal language and the natural instruction methods used by both teams (either in teaching the curricula or learning from them). This report focuses on the activities and results of the Curriculum Team.

Our BL framework includes three agents, whose interactions and relationships are shown in Figure 1. A teacher agent serves as a proxy for an eventual human teacher, instructing and testing the e-student; the automated teacher eased scaling and reproducibility for evaluation. The student agent is the embodiment of the e-student, typically employing a number of learning algorithms. The world agent serves as a proxy for a domain simulator. Over the three phases of the BL program we developed a set of laddered curricula in a variety of complex domains including Blocks World, UAV, diagnosis tasks for an ISS, ATF, planning robotic arm movements, and a hidden domain – satellite ground station monitoring and configuration. The hidden domain was kept secret from the Learning Team and used for formal testing and evaluation.

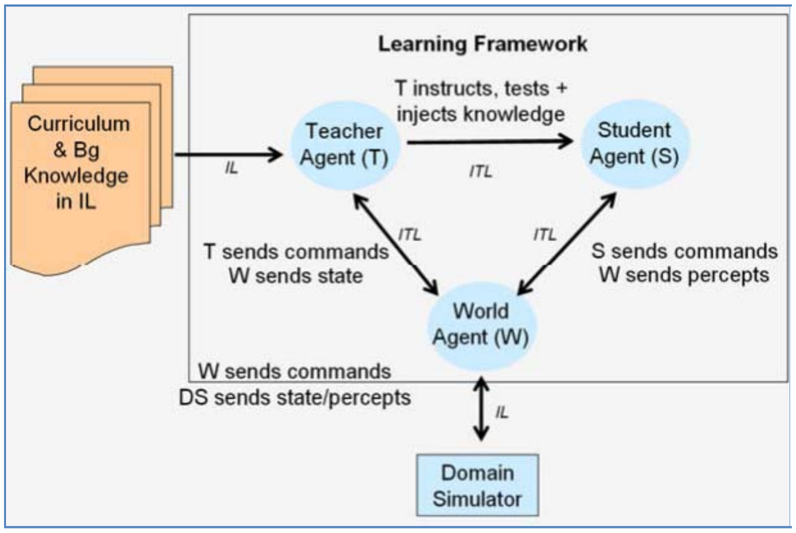


Figure 1: The Three Agents of our BL Framework

As part of the BL framework, we developed IL (InterLingua) and ITL (InteracTion Language) [5, 6] to pass messages between agents in the BL framework. The e-student used a variety of machine learning algorithms to learn how to solve tasks in an arbitrary domain, where the automated-teacher instructed with a set of formally defined NIMs. For example, in using the teaching by example NIM the instructor may make gestures at relevant problem attributes, demonstrate actions in the domain, and provide explanations for why actions were taken. Human subjects were trained and tested using a human-accessible version of the hidden domain curriculum to serve as a benchmark against which the e-student could be measured.

In addition to its core tasking (i.e., framework, curricula and e-student evaluation), the Curriculum Team conducted early experiments to explore approaches humans take for instructing others, two seedling efforts on technology complimentary to instructable computing, and completed a stand-alone BL-ISR prototype to explore and illustrate how BL might be used in practice by a human operator.

3.0 Methods, Assumptions, and Procedures

The BL program encompassed a range of experimental, empirical, evaluation and proof of concept tasks and a range of methods and procedures to achieve them. These include the following:

Software development activities:

- Definition of the core BL communication languages
- Development of the BL framework
- Development of the automated teacher

The software development aspect of the BL program was conducted primarily in the Java programming language and followed the standard BAE Systems research process model. The distinguishing feature of these development efforts was our close collaboration with the SRI International e-student team (often referred to simply as the Student Team). While the evaluation tasks in the project required a more arms-length approach, the development of software to integrate and test the e-student within the BL environment was conducted as a collaborative effort between BAE Systems and SRI International with SRI International helping BAE Systems to prioritize development tasks and with BAE Systems providing incremental releases to SRI International to facilitate their internal testing efforts.

Evaluation activities:

- Human benchmark evaluations
- E-student evaluations

The program evaluation tasks were structured to vet both the capabilities of the e-student and to calibrate the difficulty of the testing scenarios. Human benchmark evaluations were used to calibrate the test curriculum so that they were neither too hard (human could achieve a minimum of 80% success after training in the domain) nor too easy (human did not score greater than 20% before training). These test curricula were then used to test the e-student, which was required to demonstrate at least 75% of the human performance in order to be considered to have passed the tests. Testing also involved a hidden domain which the Student Team did not have access to. The potentially completely blind nature of this testing was mitigated by testing the e-student on diversity domains, access to which was shared with the Student Team. Informally, the diversity domains tested a range of learning problems including problems analogous to what would be seen in the hidden domain. The informal intuition behind this setup was that a learner that could pass the diversity domain testing should be able to pass the hidden domain testing (and this informal intuition was accompanied by an extensive analysis in Phase 2 to document the analogical similarity between diversity domains and the hidden domain).

Test data / case creation activities:

- Development of natural instruction method protocols
- Development of diversity domains and simulators
- Development of the hidden domain and simulator

Development of test data for this program involved a combination of programming-like activities and instructional material development. Natural instruction method protocols were developed to govern the testing interaction. NIMs are protocols in the computer science sense with precise definitions and they are also cognitively inspired by human-to-human instructional techniques. Testing also required development of a computation simulator for the domains under test. Test domains were rich domains with a range of objects, actions, and side effects. Manipulation of blocks is an example of a very simple domain and even that required simulators. The only domain we imagined that would not have required a simulator would have been a purely abstract domain such as mathematical performance. The final component of producing test data was the production of curriculum in the various domains. These tasks required a combination of knowledge acquisition (to learn about and define the domain) and pedagogical construction of hierarchically structured curriculum lessons (the hierarchical structuring being crucial to demonstration of the "bootstrapped" nature of the learning approach).

Proof of concept activities:

- Development of the BL-ISR demonstration system

In Phase 3 we combined the results of previous phases and our observation about what would be required to support instructional interactions with humans to produce a demonstration system of BL techniques applied in a situational instruction context to problems of ISR intelligence analysis. This activity was grounded in knowledge acquisition sessions with intelligence analysts, based on simulated Ground Moving Target Indicator (GMTI) data provided by the Air Force Research Laboratory (AFRL), and influenced by interactions with the GMTI analysis community.

In parallel to our primary BL activities, we conducted a pair of seedling efforts (Framework Adoption and PROWL) to explore approaches to system longevity, complementing the instructable computing approach taken in the BL program.

Seedling activities:

- Framework Adoption
- PROWL

The Framework Adoption problem reduces to the problem of transferring knowledge of a software framework to individual engineers. Many methods of acquiring knowledge of software exist from instruction to self-study to trial-and-error. All of the existing methods have their drawbacks and most take the engineer off-task while engaging in learning. We proposed that all these methods may be augmented by machine assistance. Our approach had two main thrusts: use machine learning to discover the implicit idioms, conventions, and best practices for the use of a software framework and provide the engineer with context-sensitive GUI assistance that provides the relevant framework knowledge just-in-time.

PROWL investigated application and extensions of social computing technologies to empower users. Under the guidance of its DARPA Program Manager it had two thrust areas. The first, the primary emphasis of our seedling effort, explored a new approach to searching large corpora of text to find relevant information exploiting a new approach to semantic markup. The end goal of a full PROWL program for Thrust 1 would be to provide automated cataloging for users to access available knowledge—data with context and understanding. The goal of our seedling effort was to demonstrate system concept feasibility. Our approach was to use the Army Knowledge Online (AKO) as a platform to demonstrate PROWL concepts. The second thrust, which represented a smaller effort, explored potential synergies between augmented reality and three social computing concepts, shared sensing, collaborative analysis, and coordinated action.

Data collected through these technologies, alone and in concert, might be used to augment data from a more standard catalog, such as the AKO used in Thrust 1. Our approach for this secondary effort was to develop a number of use cases to illustrate their potential use and impact, and to identify supporting technologies that could be used to amplify them.

4.0 Results and Discussion

4.1 BL Languages

One of the major products of the BL program is a family of languages used for communication between BL components, including agents such as e-students and electronic teachers, and components within the e-student, including learning algorithms. These languages are either dialects of or have their origins in BL IL, an object-oriented language first developed during the initial BL Seedling (executed by Cycorp before initiation of the Bootstrapped Learning program). We begin by describing that language and its role in the program, and follow with separate discussions of its notable dialects and a summary of its evolution and contributions to the program.

4.1.1 InterLingua (IL)

4.1.1.1 Overview

The BL IL is the most general language used in the BL program, and is used either directly or indirectly by every electronic agent (i.e., teacher, student, world (simulator)) in the BL framework. For example, interactions between agents are represented using the ITL, a dialect of IL, and curricula are codified as IL objects and contain sub-routines that are either IL programs or call outs to IL programs when building Interaction Language messages.

Apart from serving as a basis for these specialized dialects, to be discussed in more detail below, IL played the key role of serving as the language for representing the knowledge that the instructor intends to impart to the student. Conceptually, the instructor wishes to impart a capability to the student. The teacher, being an electronic agent acting on behalf of the instructor, has a model of this capability in the form of a procedure. The student's goal is to construct a functionally equivalent, and executable, model of this capability. In the BL program, both the teacher's model (often referred to, slightly imprecisely, as "injected knowledge," since, if necessary, it can be loaded into the student directly) and the student's learned model (often referred to as the "results of learning," or ROL) are codified in IL.

The requirement for the student to codify its ROL in IL was a major driver in IL design. First, IL supports key programming idioms, such as if/then/else and while loops, as well as declarative (logic-like) structures. IL classes are first-class objects in the language, to support the possibility of allowing a student to "reason about" the class hierarchy so that, e.g., it might sensibly estimate what IL "methods"[1] are most appropriate to try when solving a certain class of problem. Finally, IL classes can be associated with execution environments declaratively, so that the student can build its own class and decide, autonomously, how to implement semantics for that class.

4.1.1.2 Summary of Language Features

IL is a class-based, object-oriented language. Accordingly, each formula of the language can be thought of as a typed object, called an *IL object*. Each IL object falls exclusively into one of three categories: it is an *atom*, such as a string, integer, float, or symbol; a *composite*: an object

[1] In IL, the notion of a method most closely correlates to a family of executable IL classes.

formed by applying an implicit constructor for a given class to zero or more IL objects; or a *list* of IL atoms and/or composites.

The reference implementation of IL provided by the curriculum team comes with an IL parser and process virtual machine (VM), the IL VM, that enforces typing, resolves symbols to objects, and "directs" executable code to the proper interpreter.

The General Structure of IL Objects

A composite IL object can be thought of as a recursive structure, in the sense that each such object is parameterized, and the values for those various parameters are themselves IL objects. These parameters are analogous to Java class data members or to fields in a C structure. Here is an example composite IL object from Blocks World, an instantiation of the class `Block`:

```
Block(name="block1",support=Block(name="block2",color="red"))
```
[2]

This object contains two IL objects: an atom (the string `"block1"`) and one composite object, `Block(name="block2",color="red")`. The second composite, in turn, contains two atoms: `"block2"` and `"red"`. The symbols `name`, `support`, and `color` are parameters to which `"block1"`, `Block(name="block2",color="red")`, and `"red"`, and have been assigned respectively as values.

Assignment of a value to a parameter in a structure instantiation is tantamount to ascribing a property to an object For example, our instantiation of `Block` carries with it facts about the instance: It can be referred to using the symbol `block1`, and it is supported by the block `block2`, which is red.

An intended benefit of the recursive structure of IL objects is the ability to have a uniform interface to accessing IL objects through the fields of "subsuming" IL objects. This accessor is implemented in IL via a `GetField` command that takes an IL object and a symbol (a field name) and returns the value of that field. For convenience, a "`.`" operator allows for arbitrarily deep references through IL object fields. For example,

```
block1.support
```

is a reference to `block2`, while

```
block1.support.color
```

is a reference to `"red"`.

IL Classes

A notable feature of IL is that IL classes can be declared piecemeal. The `Is` operator allows one to define a new class and place it in the class hierarchy, or to take an existing class and place it underneath multiple classes, so as to effect multiple-inheritance. For example, given the existence of the class `Animal`, one can define a new class `Person` as a sub-class:

[2] This example is designed to illustrate the way in which IL objects can structurally subsume one another. The example does not cover all possible forms of IL objects or valid ways of writing IL; see "IL Parsing and Printing" [7] for a full specification.

```
Is(Person,Animal);
```

One can then define `Person` as a sub-class of `Intelligent`:

```
Is(Person,Intelligent);
```

This enables the `Person` class to inherit parameters from both classes.

Parameters can also be explicitly defined for a new class, again in piecemeal fashion, via the operator arg. For example,

```
Arg(Person,almaMater,School);
```

defines the `almaMater` parameter for `Person`, and constrains its values to objects that instantiate the `School` class. This means that any valid instantiation of `Person` can be assigned an `almaMater` of the appropriate type, in addition to any parameter defined for its super-classes. For example, assuming that `Animal` has a parameter, `ageInYears`, that takes an integer, one can instantiate `Person` with an IL object:

```
Person(name=Bob,ageInYears=32,almaMater=UniversityOfMinnesota)
```

IL Evaluation

For some IL classes—specifically, those that represent executable commands or functions—it is desirable to assign an interpreter that can be used to execute instances. Again, this assignment can be done declaratively in IL, using the `defCode` operator. `defCode` requires three arguments: a class name (the class whose instances will be evaluated by the interpreter), a pointer to an engine, and a "chunk" of code. For example,

```
defCode Plus Native NativeBody(code=Arithmetic);
```

When an actual argument list, e.g., `Plus(2,3)` is interpreted, the engine behind the `Native` engine pointer looks up the relevant code in the code chunk pointed to by `Arithmetic`. That engine runs its code on the values `1` and `2` and updates the evaluation context with a new `Number` construct with its result. Thus, for example, `Plus(2,3)` evaluates to an invocation of addition (i.e., `Plus` instantiated with arguments `2` and `3`):

```
Plus(args=2,3,name="plus-344",returnValue=5).
```

The astute reader will observe that in this example, the result of executing `Plus` here is an updated `Plus` object that carries with it its `returnValue`, and not the value itself (`5`). This perhaps unusual feature of the language was intended to support learning, by minimizing information loss (specifically, by generating a record the student can inspect of what functions were called and which operations were executed. This supports student "experimentation" with code during 'by Feedback'-style instruction. Over the course of the BL program, however, this level of information preservation was deemed to be less useful to the student than had been anticipated, and a cause for slowness within the BL framework. As a result, this feature was eliminated.

In the example `defCode` above, we call out to a pre-defined chunk of code that is supported by the `Native` engine. However, IL was designed to allow its users, including (especially) the e-student, to write arbitrary code. For example, the code for the `CalcHypot` class could be represented in the following `defCode` declaration:

```
defCode CalcHypot Function
          FunctionBody(expression=Sqrt(Plus(Expt(a,2),Expt(b,2))));
```

In the typical case, the e-student was expected to learn `defCode` declarations that were compositional in this way, building upon previously learned `defCode` declarations and adjustable by various learning algorithms within the student.

IL Packages

All IL objects, including class and code declarations, are relativized to a specialized object known as a `Package`. `Package` objects can be thought of as “bundles” of declarations organized into a hierarchy. A package “below” another package inherits the contents (the objects) of the “higher” package. This sort of inheritance can be illustrated by thinking about how packages affect the ability of an IL VM to dereference the names of IL objects: if `Package-1` contains the object `Object(name=obj1)`, and if `Package-2` contains the object `Object(name=obj2)`, and `Package-3` inherits from `Package-1` and `Package-2`, then the following claims about the names ‘`obj1`’ and ‘`obj2`’ can be made:

- In `Package-3`, the names ‘`obj1`’ and ‘`obj2`’ will be recognized as names of objects, and parsed to `Object(name=obj1)` and `Object(name=obj2)`, respectively.
- In `Package-1`, the name ‘`obj1`’ will be recognized and parsed to `Object(name=obj1)`. The name ‘`obj2`’ will not parse to an object, and will be understood as a bare symbol, `obj2`.
- In `Package-2`, the name ‘`obj2`’ will be recognized and parsed to `Object(name=obj2)`. The name ‘`obj1`’ will not parse to an object, and will be understood as a bare symbol, `obj1`.

The purpose behind the package mechanism was to enable the e-student to formulate competing hypotheses about the concept it is learning. That is, as a result of instruction, the student might have two possible interpretations of the semantics of some concept (i.e., two competing `defCode` declarations), and the package mechanism allows it to maintain both without contradiction. A secondary benefit of packages was to support the implementation of injected knowledge – that is, if the student needed to be “handed” the teacher’s interpretation of the target concept to be taught (a strategy designed to allow the student to be imperfect, and “fail” an intermediate test yet still “carry on” without necessarily failing the entire curriculum). This could be accomplished simply by given the student access to the teacher’s package for that knowledge.[3]

As noted above, in instruction as viewed in BL one can think of there being a target capability (or “concept” to use BL vernacular) that the instructor intends to impart to the student. The teacher has its own internal model, or codification of this capability represented as a series of `Is`, `Arg`, and `defCode` declarations. Rather than just “handing over” this internal model to the student, the programmatic goal of BL was to produce a student that could build a functionally equivalent model through instruction. This requires a language for communicating between the

[3] Originally, this was accomplished by simply adding a package inheritance declaration, making this an elegant solution to the Injected Knowledge problem. However, as the program moved towards a fully distributed system (with the teacher and student running on different VMs), the sharing of knowledge was

accomplished by serializing the package. student and teacher, and for communicating facts about the shared, simulated environment and changes to that environment. It is to this language, the InteracTion Language, which we now turn.

4.1.2 InteracTion Language (ITL)

ITL is a dialect of IL used for inter-agent communication. For details on ITL the reader is directed to the BAE Systems Technical Report "Bootstrapped Learning Interaction Language" [5], the canonical documentation source for ITL. Here we provide an overview.

Though the ITL language contains many constructs – many of which are specific to Natural Instruction Methods (Section 4.2) – at the most general level of description, the language consists of four main classes. These classes serve to bundle up information about the simulated environment or to express "spoken" forms of communication between agents. These classes are all extensions of a top level ITL class, `Message`:

- `Perception` messages contain information about what an agent can currently discern about objects in the simulated world, such as their relative location.
- `Action` messages reflect "atomic" activities, such as which simulator commands have been executed, that can be presumed to be detectible by an electronic agent. (These are distinguished from full-blown tasks that are implemented, in part, by the execution of simulator commands – such tasks are not directly perceivable, and so are not "given" in `Action` messages.) To disambiguate the doer of an `Action`, the class is actually sub-classed into specialized `TeacherAction` and `StudentAction` classes, specific to the teacher and the student, respectively.
- `Utter` messages allow agents to express truth-evaluable claims about the world.
- `Imperative` messages allow agents to give commands to one another.

These classes all inherit `source` and `addressee` parameters from `Message`, which allows the BL framework to direct messages to their appropriate recipient. Together, these message-types can generate a useful model of teacher-student-world interactions. Consider a brief example of a fictitious interaction sequence from Blocks World:

The Blocks World simulation starts and all agents receive a `Perception` of the initial world state, a table with a block, `a`, on the table, and a claw:

```
Perception(
  perceptsGained=
   [Table(name=table),Block(name=a,support=table),Claw(name=claw,holds=NIL)],
  perceptsLost=NIL)
```

The teacher then begins to execute a simple "lift block" routine. This is accomplished by sending a request, in the form of an `Imperative` message, to the simulator. This message, addressed only the simulator proxy agent called "`theWorld`," is not visible to the student.

```
Imperative(
  addressee=theWorld,
  source=theTeacher,
  request=LiftBlock(block=a))
```

In response, the simulator executes the first of the two low-level actions of the routine, which are reported as `Action` messages credited to the teacher:

```
TeacherAction(
  action=Grasp(grasped=a))
```

As a result, the simulated "world state" changes: `a` is now in the clutches of the claw:

```
Perception(
  perceptsGained=[Claw(holds=a)],
  perceptsLost=[Claw(holds=NIL)])
```

The teacher then articulates this fact to the student (who, we'll suppose, is being taught what "in" means in this context, and for which this state is an example from which the student can learn):

```
Utter(
  addressee=the-student,
  source=theTeacher,
  utterance=In(a,claw))
```

The simulator next executes the second of the two low-level actions that make up the "lift block" routine, reporting this back to the timeline as a teacher action:

```
TeacherAction(
  action=Raise())
```

The world state changes, and again the teacher comments on this change:

```
Perception(
  perceptsLost=
    [Block(name=a,support=table)],
  perceptsGained=
    [Block(name=a,support=NIL)])

Utter(
  addressee=the-student,
  source=theTeacher,
  utterance=Raised(raised=a))
```

The relative temporal ordering of messages is modeled by their order on a `Timeline` object that is generated for each instructional sequence. Additionally, messages can be assigned a `timestamp` to generate more concrete temporal information.

4.1.3 Curriculum Language[4] (CL)

CL is used to codify instructional materials for consumption by the student in the BLADE framework. At the most general level, a codified curriculum consists in the following set of objects:

[4] For detailed information about CL, the reader is directed to "Bootstrapped Learning Curriculum Language" [8].

- **Injected Knowledge**, which consists of the teacher's IL formalization of the capability (or "concept") to be taught.
- **Generators**, which are executable programs that generate sequences of `Message` objects (see the section on ITL, above) for consumption by the simulator and student. Generators come in two varieties, **Static Segment Generators**, which play a fixed sequence of messages, and **Message Generators**, which may produce different message-sequences in response to changes in/reactions from the world state/student.
- **Lessons**, which invoke generators to produce a sequence of messages.
- **Tests**, which pose problems (see the discussion of `Imperative` messages above) for the student to solve, for the purpose of evaluating whether the student has acquired a functionally equivalent representation of the target capability
- The **Curriculum** itself – an object that associates lessons and tests with their target concepts (the Injected Knowledge) and orders those concepts in a hierarchy, based on which concepts are prerequisites for others. Within the program, a metaphor of a ladder was used to describe this hierarchy.

Figure 2 reproduces a diagram from the "BL Curriculum Language" documentation that draws upon the ladder metaphor to illustrate the general structure of BL curricula. IL programs called segment generators produce sequences of messages from the teacher that form the basis of instruction. A lesson is a structure that contains one or more segment generators; each lesson teaches a concept (or rung), which may have several lessons. A curriculum is a sequence of rungs ordered by a prerequisite relation (a *higher* rung requires, or makes use of, knowledge taught at a *lower* rung).

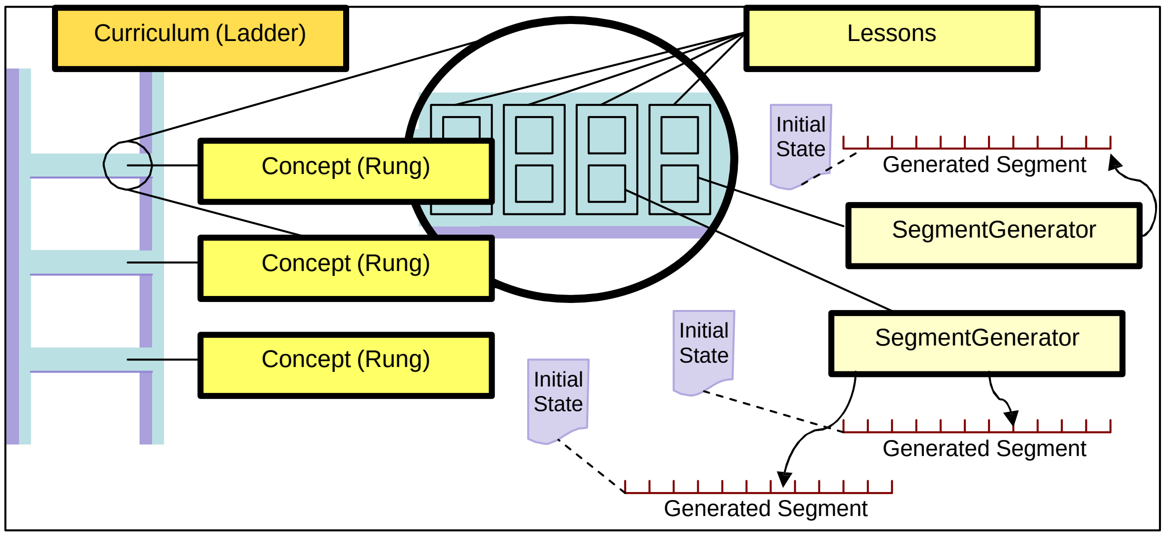


Figure 2: Structure of a BLCurriculum

In the early stages of the BL program, the Curriculum Language was, like the Interaction Language, an IL dialect; segment generators, for example, were executable IL objects that were executed in an IL virtual machine within the BL framework. The decision to implement the CL in IL had the drawback of requiring IL expertise, so that only a handful of developers could actually contribute to the process of curriculum formalization. At the same time, this decision had the benefit of turning curriculum authoring into an intensive IL proving ground; much of the work that went into stabilizing and maturing IL was prompted by early curriculum development efforts, early enough that most IL bugs were discovered internally by the curriculum team, and not "too late" by the Student Team developers.

As the language matured and IL stabilized, this benefit no longer outweighed the bar to entry in curriculum development that IL expertise represented. Thus, in Phases 2 and 3, alternative methods of implementing generators were developed, first in Java, later in Ruby. Both of these alternatives lowered the qualifications bar and potentially opened up curriculum authoring to people outside the immediate BL community.

4.1.4 Language Evolution and Implementation

Prior to the start of the BL program, a reference implementation of IL had been developed, to allow the research program to hit the ground running. When the BL program began, this implementation, the IL Virtual Machine, was published as part of the BL framework and made available to all program participants. However, the Student Team developed its own implementation of the language, to support aspects of the student design, which required that the various internal learning components communicate with one another in IL. After much negotiation, the Student Team developed IL 2, a second implementation that met the Student Team's requirements while simultaneously allowing the curriculum authors, who had a high level of facility with the original IL, to continue to write curricula in accordance with original IL assumptions.

Though IL did not undergo any other major changes after this, it was extended during Phase 3 to support reactivity. Reactivity began to emerge as an implicit requirement for some of the curricula under development in Phase 2, such as ISS and the Hidden Domain, where the metaphor of a "monitoring agent" was used to describe the desired post-learning behavior of the student. However, IL had not been set up to handle reactivity – the instructional model was that the student would only do what it had been explicitly told to do via an Imperative message – nor was its single-threaded implementation designed to handle complex issues such as the concurrency that reactivity presupposed.

By the end of Phase 3, the language had been extended to simulate a restricted form of reactivity, via an **`Anytime`** construct. An object of the form **`Anytime(condition=<TRIGGER>, action=<RESPONSE>)`** when registered in the VM would enable a sub-routine whereby the student would check whether the predicate **`<TRIGGER>`** evaluated to true each time the world state updated, and when it did evaluate to true, it would cause the program **`<RESPONSE>`** to execute. To support this kind of reactivity, the BL framework was also extended to provide a world state server that could reliably maintain a directory of all percepts in the simulated environment at a given time.

This work on the world state server allowed the curriculum to also support historical state references when developing Phase 3 curricula. That is, while in the first two phases, the instructor could only refer to objects in the "current" world state, the ability to access a world state server allowed the instructor to make, and for the student to understand, references to changes in state, to facts that had been true in the past but no longer were. This extension played a role in ensuring that Phase 3 training and evaluation curricula were richer in the space of problems to be solved.

Other changes to IL have been alluded to above, but are worth mentioning here: The preservation of all IL objects, including invocation records of executed programs, was eliminated in favor of a more traditional approach of maintaining only the return values. Additionally, explicit quoting support was also introduced as a valuable way to delay the execution of IL objects, so that the student could inspect an object (e.g., a teacher's utterance about the world state) without evaluating it (and winding up with 'true' instead of the utterance itself).

4.1.5 Uses Outside of the BL Framework

As noted above, the Student Team implemented its own version of IL in Java during Phase I for use as a communication language among learning components. Internally, this involved extending the IL language to support a First Order Logic (FOL) for describing the structure of IL objects. For example, given an object of the form `Tank(name=tank712,moving=true)`, the student would generate an additional object describing this form, using IL predicates to state, for example, that the value of the `moving` parameter for `tank712` is `true`. This use of IL played a large role in the byExample and byFeedback learning algorithms within the MABLE student; these algorithms would produce `defCode` declarations that targeted a specific prolog engine, and would compose a code chunk that drew upon this FOL representation.

4.2 Natural Instruction Methods (NIMs)

Natural Instruction Methods (NIMs) are protocols for interaction between a human or electronic instructor and an e-student. Within the BL program, these protocols were intended to serve two functions: First, they defined the "legal" space and ordering of messages that comprise the instructional input to the e-student over the course of a lesson. In this capacity, NIMs provide a rigid infrastructure that insulates learning algorithms from exposure to "unfair" or "trick" interactions, and thus facilitates establishment of a stable development environment for the learning team and supports a fair evaluation of the learning team's algorithms. Secondarily, NIMs have served as models of human-to-human pedagogical interaction. In this capacity, they were intended to help "drive the science" of BL forward, by introducing challenges that human students can surmount, but the state of the art in machine learning could not. Though not in direct conflict, the two roles are in tension, as the more constrained and "stable" the interactive protocols, the less faithful to natural human interaction the model is. This tension was intended to be alleviated by modifying NIMs over time, so that once learning algorithms had been proved robust against a set of initial, admittedly artificial protocols, those protocols could be changed in piecemeal fashion to accommodate the sorts of problematic features – such as ambiguous or vague use of language and the omission of "obvious" pieces of information – that are ubiquitous in human instruction. This transitional process was termed "relaxation," and was central to supporting the program's goal of developing learning algorithms that would be of enduring value to the instructable computing field.

In this section, we describe the kinds of NIMs that were developed and deployed over the course of the BL program. We discuss relative successes and failures in the transitional "relaxation" process, and assess the utility of the corresponding NIMs to the program and to the instructable computing field. Additional documentation can be found in BAE Systems Technical Report 2766 [9].

4.2.1 NIM Types

As will be discussed, some of the initial assumptions about the nature of NIMs (that they weren't specific to learning algorithms) did not survive the early phases of the BL program, somewhat reducing the overall utility of NIMs as tools for promoting learning algorithm development.

4.2.1.1 Identification of Methods and Specific Knowledge-Types

As formalized protocols, NIMs are best thought of as specifications that fall most generally into three general categories corresponding to methods of teaching, but within each category, different NIMs specify variations in the protocol based on the type of knowledge being taught. The three general NIM categories recognized by the BL program are the following:

- Teaching by examples or demonstration ("by Example" for short)
- Teaching by telling or describing ("by Telling")
- Teaching by feedback ("by Feedback")

As the program developed, it was recognized that these general NIM categories apply to several different types of knowledge:

- Procedural, or "how to" knowledge, of which two subclasses were eventually recognized:
    - Procedures that result in perceptual changes in world state (e.g., making a stack of blocks), and
    - Procedures that calculate values for (non-Boolean) functions.
- Knowledge of conditions, of which three relevant types were recognized:
    - Truth-conditions (i.e., the semantics of a predicate were taught)
    - Execution conditions (i.e., predicate(s) to check whether an if/then/else construct or while loop should be entered)
    - Post-conditions (i.e., how the world changes as a result of executing some procedure)
- Syntactic knowledge, or knowledge of the definitions that define new classes, where in the class hierarchy those classes fall, and the parameter/value-typing signatures that define the well-formedness conditions for their instances

The identification of knowledge types was driven primarily by three factors:

1) *Momentum from the BL seedling*: Many of the kinds of problems taught in Blocks-World fell into obvious, natural categories (e.g., predicates, procedures, post-conditions, syntax), and these were more or less accepted as "good knowledge-types" by program participants
2) *The initial design of curricula by domain providers*: The curriculum team had domain providers who supplied simulators, domain expertise, and informal curriculum designs that were to be codified by other members of the team. As these informal designs developed, certain kinds of knowledge implicit in the designs became evident.
3) *Design decisions by the learning team*: The learning team developed an e-student architecture that assumed the existence of a central control module whose job was to identify – at the level of individual lessons – which learning algorithm(s) should be applied. Since, at least initially, each lesson taught one concept (knowledge of some type) and used one general method, the assumption that individual lessons could be handled entirely by a single learning algorithm meant that learning algorithms were to be defined by general method and knowledge-type – i.e., by NIM. This gave learning algorithm developers – machine learning experts with an interest in specific learning

problems – a genuine stake in what kinds of things were being taught and how, and this influenced the kinds of knowledge that would be recognized as falling under a formalized NIM protocol. Thus, for example when "procedural knowledge" was to be taught by Example, it was important that a distinction be drawn between "procedures with side effects" and "procedures that calculate values for functions," because two different learning-algorithm writers were interested in, and responsible for, these two types of procedures. This distinction did not need to be explicitly made when teaching by Telling, however, as a single algorithm-writer was responsible for learning procedures via Telling.

4.2.1.2 The Development and Refinement of NIM Contracts

Early on in the program, each type of knowledge was associated with 3 separate NIM protocol formalizations, one for each general method (the sole exception to this was Syntactic knowledge, which was recognized as only sensibly taught by Telling). These formalizations underwent a process of generalization, combining protocols into a unified formalization where possible. By December 2008 (the middle of Phase 1), this process of generalization was complete, and the specific knowledge-types were officially combined with the general method-types to define eight sets of NIM protocols, known as *NIM Contracts*:

- ***TellingOfProcedure*** *– both types of procedural knowledge, taught by Telling*
- ***ExampleOfProcedure*** *– both types of procedural knowledge, taught by Demonstration*
- ***TellingOfConditions*** *– all three types of condition, taught by Telling*
- ***TruthConditionsByExample*** *– truth conditions, via positive and negative cases*
- ***ExecutionConditionsByExample*** *– if/then/else, while triggers, by positive/negative cases*
- ***EffectsByExample*** *– post-conditions, by positive/negative cases*
- ***byFeedback*** *– all knowledge-types (except Syntax), taught by testing and grading*
- ***TellingOfSyntax*** *– telling of classes and argument-signatures*

The NIM contracts were delivered to the learning team, who delivered them to the appropriate algorithm-writers. In addition to the contracts, a "universal" protocol document that defined commonalities – the "NIM Common Terms" document – was developed and distributed. The NIM Common Terms document dealt with protocols that were shared among NIM contracts. For example, every NIM contract specified that lessons begin with "control utterances" – a pair of statements from the instructor that identified, first, the type of concept being taught, and then the general method, so that the e-student would know what NIM protocol applied. For example, to indicate that the ***TruthConditionsByExample*** NIM contract was guiding the current lesson to teach a predicate named "`Foo`," the instructor would produce the following pair of control utterances:

```
LessonTeaches(WhenTrue(Foo))
Method(byExample)
```

In contrast, if the ***byFeedback*** contract was guiding the current lesson, which taught `Bar`, a procedure (with side effects), the instructor would produce this pair of control utterances:

```
LessonTeaches(HowTo(Do(Bar)))
Method(byFeedback)
```

The NIM Common Terms protocol also dealt with other commonalities, such as the use of so-called "relevance" vocabulary across the ***byFeedback*** and the various ****byExample*** NIMs, the definitions of logical vocabulary (`And`, `Or`, etc.), the use of language to specify desired end-states, etc.

4.2.1.3 Contributions from the Learning Team

Early on, the learning team did not actively attempt to contribute to the formation of NIM contracts, opting for the most part to study the blocks world curriculum and implement algorithms that could target those lessons as use cases. Indeed, early on, some members of the learning team questioned the value of NIM contracts altogether, suggesting that progress would be obtained more rapidly if the development of training curricula were accelerated relative to development of formalized protocols. But as diversity domain curricula began to emerge, and the blocks-world use cases became insufficient to predict the space of interactions included in the new curricula, the need to attend to and help shape the details of the NIM contracts became clearer to the learning team, which took two courses of action in response. One was to produce a document on “learnability,” in which each learning algorithm writer enumerated the general features required by any lesson teaching a concept falling within that algorithm’s bailiwick. The second was to engage with the curriculum team directly, identifying particular problematic examples from the diversity domain and extrapolating from those examples what sorts of changes needed to be made, either to the NIM protocol or to the learning algorithm. Working with the learning algorithm writers individually, several changes in NIM contracts were introduced, some in the area of additional vocabulary to facilitate expressiveness and reduce ambiguity, others in the area of restrictions that would increase clarity. One of the most notable improvements came from discussions that began during the learning team-hosted “hackathons,” where extensions to the then-generic “relevance” language were developed. (Up to that point, there had been a single predicate, Relevant(X) which meant that the object X was somehow relevant to what was being taught, with no indication about how or why the object was relevant). Other contributions were more restrictive, such as eliminating multiple ways to say the same thing, and eliminating ambiguity. Most of these restrictions attained the status of relaxation trajectories, or planned “loosenings” of the now stricter protocols, that could be used later in the program.

4.2.1.4 Syntax and Noticing

Of the eight formal NIM contracts used in Phase I of the BL program, seven continued to be used throughout the remainder of the program. No additional contracts were added. The ***TellingOfSyntax*** NIM contract was eliminated at the end of Phase I, and replaced by the learning strategy of *Noticing* that new terms had been introduced in the course of a lesson, and deriving the IL definitions by observing how the terms were used. This added a new dimension of realism to the diversity domain curricula, as it allowed an entirely new concept to be introduced and its semantics taught in the course of a lesson.

4.2.2 Assessing the NIMs

4.2.2.1 Phase I & II NIM Assessment

It is fair to say that the NIM contracts of Phase I – and thus the diversity domain and Phase II Hidden Domain lessons that implemented them – successfully filled their role as formalized protocols to guide interactions and ensure that concepts were learnable by the e-student. The e-student passed the pre-Hidden Domain acceptance test during Phase II on the diversity domain lessons, each of which instantiated a NIM contract, and the breadth of which exercised the breadth of “instructional space” allotted by the relevant NIM fairly well. Additionally, the e-student passed the Phase II Hidden Domain curriculum, which established that the NIM protocols and the learning algorithms were robust when presented with a novel domain.

At the same time, however, the initial versions of the NIM protocols were highly restrictive by design, and so were fairly artificial, limiting their ability to serve as accurate models of human instruction. Somewhat unfortunately, the NIM contracts – influenced by largely restrictive contributions from the learning team that came relatively late in the first phase of the program (only after diversity domain lessons became available) – arguably took a step backward from their role as models of human instruction, becoming more artificial than what had been initially proposed. Areas of artificiality thus introduced included:

- An increased level of "relevance" annotation of examples (providing more information than humans would plausibly provide) that caused most "by Example" and "by Feedback" learning to approximate the "by Telling" interaction protocol.
- A need to explicitly introduce, as arguments to a predicate, any domain object used in calculating its truth value. (e.g., instead of defining "siblings(x,y)" as "given x,y, and z, siblings(x,y) holds if parent(x,z) and parent(y,z)", the restriction was introduced that siblings be defined as ternary, with an argument place for the parent.)
- The elimination of disjunctive truth conditions as possible learning targets.
- A reliance on explicit statements of "what to do next" while demonstrating a procedure, e.g., that a procedure contains a loop or an if/then/else construct.

4.2.2.2 NIM Relaxation and Phase III

After Phase II, the curriculum team took measures to introduce a space of learning challenges and relaxation trajectories to reduce NIM artificiality and move towards more realistic modeling of human instruction. In March 2010, the curriculum team distributed a document to the learning team defining a fairly rich space of such challenges and relaxations that would improve instruction naturalness. The learning team was able to implement some of these new NIM types; fortunately, some of the most troubling artificialities "made the list," and much of Phase III was spent implementing these relaxations in the diversity and Hidden Domains. For example, all of the bulleted items, listed above as matters of concern with respect to artificiality, except for "disjunctive truth conditions" were implemented in Phase III and used by the algorithm writers to allow their learning algorithms to function with less artificial NIM protocols.

By the end of the Phase, the e-student was able to reach performance on the "relaxed" diversity domains comparable to that in Phase II, and it succeeded in passing the "relaxed" Hidden Domain. Though some artificialities remain, the fact that the student was able to learn with several artificial "crutches" removed, and with only a short time for additional preparation by the learning team is highly promising.

4.2.3 Conclusions

Natural Instruction Methods were an integral part of the BL program, serving both as protocols to ensure compliance of instructional materials with the expectations of the learning team and with the requirements of fair testing, and as models of human modes of pedagogical interaction. While highly successful in this first role, the NIM protocols developed for the BL program were only able to begin to live up to their second – arguably more important – role in the final phase of the program. Nevertheless, the fact that the e-student was able to pass the final evaluation, a "hidden domain" curriculum that embodied several NIM relaxations, including the removal of excessive “what to do next” hints during demonstrations and the use of vague or ambiguous “relevance” statements, as well as multiple learning challenges, indicates that the basic approach of specifying instructional protocols in the form of NIMs has value, and can be built upon in future work on instructable computing.

4.3 BL Automatic Teacher and Framework

The e-student is taught and tested via a machine readable curriculum definition (described in a previous section). Curricula have been encoded for each of the testing domains (described in the following sections). The curriculum is transmitted via the automated teacher to the student in a manner that obeys the natural instruction methods. While a human teacher might also provide this curriculum-based instruction, in this project, the automated teacher has the role as an electronic proxy for the human instructor who writes the curriculum. While the ultimate goal of the BL program is to allow humans to instruct the automated learners, the automated teacher and curriculum combination is vital to executing repeatable and automated tests of the e-student. These testing cycles are crucial to both the development of the learning capabilities (not surprisingly the student algorithms do not learn everything required upon first release) and to the assessment of the learners’ accomplishments.

The teacher operates using the services of the BL framework. The BL Messaging Framework is a Java-based platform for conducting machine learning experiments. The heart of the framework is the Channel Server, an object which acts as a message bus and coordinates the initialization of the system. The Channel Server allows for the passing of Channel Messages on a variety of named Channels. Channels record and deliver Channel Messages to various subscribers. Channel Listeners are objects that subscribe to various named channels in order to receive Channel Messages. Agents are objects that, after registering with the Channel Server, are allowed to post as well as receive Channel Messages. In order to abstract Agents from implementation details of the underlying message bus, Agents use a Channel Message Factory to create Channel Messages for posting to the framework.

The framework has been designed so that components may be initialized and started independently from other components in the system. The first component to begin operation must be the Channel Server. The Channel Server starts a Framework Agent, which represents the framework when passing messages on named channels. The Framework Agent posts framework properties, announces agent registrations, and transmits curriculum files to agents. After the Channel Server is running, Agents and Listeners may connect to the framework at any time. Channel Listeners connect to the system by subscribing to a named channel. Agents connect to the system by registering themselves. The Channel Server remains running regardless of Agent activity. On the event that the Channel Server should shut down, it will send a shutdown message to all agents in the system.

A crucial aspect of the framework is the timeline. It can be thought of as the communication backbone for the BL framework and constituent components (i.e., Student, Teacher, and World). The timeline provides:

1) Agent ITL messaging: Agents must have an interface to post messages to other agents and to receive messages from other agents.

2) Historical access to the ITL messaging sequence: Agents must have a way to access previous messages on their timeline (a sequential accessor, e.g., getNth would provide the required access).
3) Access to the changing world model: At any particular point on the timeline there is an associated percept state of the World. An agent should be able to retrieve the percepts state of the world as of a particular time (as denoted by a particular message number in the timeline).
4) A messaging/addressing/interoperability backbone: The messaging framework is the primary interoperability mechanism that joins all the agents together. The timeline implementation should be pluggable to support alternate interoperability mechanisms. (This not so much an API requirement as a design/implementation constraint.)
5) The timeline supports flexible addressing of messages to multiple agents. Initially we have 3 addressable agents: Student, Teacher, and World. In the long run we may need to address multiple teachers and multiple students.
6) Agent coordination backbone. The timeline will also host agent coordination messages such as StudentDone and TeacherDone. The current supported instructional protocol involves a handshaking between Teacher and Student. The current protocol assumes a turn taking interaction between the agents. This turn taking does not provide for one agent interrupting another. A future requirement is for the teacher to be able to "interrupt" the student and tell the student they are done ("pencils down" – so to speak). Supporting an interruptible agent messaging protocol implies a corresponding requirement on the agents, i.e., to be able to gracefully handle an interrupt and presumably to be able to produce partial results at the point of interruption.

The framework provides programmatic coordination of the agents in the BL system. The framework provides:

1) A general mechanism for setting and communicating execution input parameters. There are mechanisms for setting general framework parameters and individual agent parameters. This parameter mechanism is generalized in the sense there is not a specific a priori set of parameters. There is a dynamic capability to register and lookup parameter names.

   Example parameters include: How many knowledge injections can the student take? How much clock time is the student allowed? (Note that multiple agents will need to be able to handle these parameters. For example, does the student have the property that it can return partial results after a fixed amount of time? If not, the clock time parameter will potentially result in the student not being able to produce any result in the allotted time.) As parameters are defined, the required processing of that parameter by each agent must be considered.
2) A general mechanism to capture agent component output data. The parameter mechanism serves as both input and output mechanism. Certain results of a learning session are returned in output parameters. This helps support programmatic invocation of learning sessions (a la UNIX pipes).
3) The APIs for running a set of agents through a curriculum. The framework provides a set of APIs for marshaling the required set of agents (Student, Teacher, and World) and running a curriculum through them. The base curriculum to be run may be customized or

modified dynamically. A mechanism of "Curriculum Deltas" provides this functionality. A curriculum delta is a specification that can be applied to an existing curriculum to dynamically define a derived curriculum. The delta may, e.g., override properties defined in the curriculum IL files.

4) The APIs for programmatically executing an individual lesson. An API is provided for running a single lesson (this might be achievable via a curriculum delta) to support the assumed common testing case of scripting a sequence of lesson invocations programmatically. The API for running a single lesson exposes hooks for the basic steps of running a lesson – including at a minimum before and after lesson run hooks

The automated teacher makes use of the framework to interact with the e-student. Curriculum segment materials are combined by the electronic teacher to create a segment generator for the lesson. The electronic teacher uses these materials to create an initial state for the domain simulator, adds messages to the timeline for the instruction, observes the student's additions to the timeline, provides feedback and evaluation when appropriate in the lesson, and answers student questions. Our baseline electronic teacher, a simple scripting language with rule support, interprets the curriculum to generate interactions with the e-student that are recorded on the timeline. For example, in a teaching-by-demonstration example, the teacher first adds scaffolding utterances and task information to the timeline. It then issues imperative utterances that when evaluated cause actions to be taken in the simulator which in turn cause a response consisting of percepts that are added to the timeline. During this interaction, the Teacher monitors the interpreter's execution and inserts utterances at appropriate points as directed by the curriculum materials. Context-specific questions, teaching-by-feedback, and evaluation rules can be handled similarly.

4.4 Diversity Domains (and Simulators)

Diversity domain curricula, or "diversity domains" for short, are electronic curricula used as development environments for learning algorithm development. Each diversity domain requires interaction with a domain-specific simulator, and is comprised of rungs that are intended to challenge the e-student on a variety of learning problems, each salient for that domain. Domains are topically heterogeneous – a gate against over-fitting of learning algorithms against a small space of problems. The problems instantiated in the diversity domain curricula were determined by a multi-phase process, involving members of both the curriculum and learning teams. The ultimate determination of learning challenges was but one step in the overall domain development process:

- ***Informal Design and Simulator Development***. First, a domain provider produces an informal description of a domain, or topic (e.g., problems in UAV control), and proposes a number of specific concepts to teach in that domain. The domain provider also provides a simulator and an API for executing commands in that simulator.
- ***Simulator Integration and Initial Formalization***. Other members of the curriculum team were responsible for integrating the simulator into the BL framework, which involved defining a "percept model" for generating ITL perception messages suitable for consumption by the student and teacher, and for codifying the concepts informally specified by the domain provider. Curriculum formalization occurred within the formalization guidelines set forth in the NIM contracts, and was constrained by the

practical expressive limitations of the IL language. As domain providers, in general, were neither familiar with the finer-grained details of the NIM protocols, nor with the practical limitations of the Interlingua language, this meant that the codification process often involved some iteration between domain providers and curriculum implementers, so that the original intent behind the curriculum was not lost or skewed as a result of formalization. Because curricula were designed in a modular fashion (that is, concepts and lessons could be added or removed without negatively impacting the ability to execute a curriculum), curricula could be "rolled out" to the learning team in piecemeal fashion. That is, once a lesson from the formalized curriculum was able to pass unit (JUnit) tests that ensured both the syntactic correctness of the lesson and the correctness of the "injected knowledge" (the instructor's formalized model of the concept, or target capability, being taught by a lesson), the lesson was made available to the learning team. This sort of "roll out" allowed for a third, final stage in the curriculum development process.

- ***Iterative Refinement***. A diversity domain curriculum was not "set in stone" until the learning team was able to do some initial experimentation, which often revealed a number of issues. Issues ranged from purely "curriculum-side" problems, such as code- level bugs that were not "catchable" by the injected student (for example, the teacher might label an example incorrectly) in JUnit testing; to inter-team interpretative issues about what the NIM protocols licensed or did not license; to "student-side" issues, such as limitations on what kinds of challenges were "fair" (regardless of what NIMs did or did not license, or the revelation that extra-NIM restrictions must be imposed for certain learning algorithms to gain traction).

This process produced four diversity domain curricula, plus an ever-evolving Blocks World curriculum, that the learning team was able to use in experimentation and algorithm development. As algorithms matured, diversity domain curricula were modified to make existing learning challenges more difficult (the NIM "relaxation" process), or to add new learning challenges. In fact, one of the diversity domain curricula – the ISS2 curriculum of Phase 3 – was written entirely for the purpose of handling new learning challenges, something that its simulator was uniquely suited among diversity domain simulators to support.

In what follows, we describe each of the diversity domains (and Blocks World), to give the reader a sense of the kinds of learning problems to which the e-student was exposed. A much more detailed description of each diversity domain curriculum appears in the documentation associated with each curriculum [10]; the reader is specifically directed to the curriculum design document, simulator design document, and phase-specific editions of the curriculum manual for each diversity domain curriculum.

4.4.1 Blocks World

The Blocks World "starter" curriculum continued to evolve with the BL program, and was used to introduce new problems during each program phase. As its name implies, the curriculum involved the ability to use a claw to manipulate a set of blocks (located on a table) to create designated block configurations, e.g., a stack. However, the Blocks World curriculum was central to the program during Phase 1, when a subset of its problems formed the basis of the Phase 1 evaluation. The e-student had to learn three target concepts from Blocks World, using three different NIM protocols, so as to establish the feasibility of their approach to learning.

In Phase 2, Blocks World was updated to use a new simulator that supported differently-sized blocks as well as absolute (x,y) coordinate locations; the original simulator only supported relative locations, e.g., “on a block” or “on the table.” The new simulator allowed for a range of more complex procedures to be taught, such as making a doorway or a T-shaped block structure.

Blocks World also proved useful in Phase 3, as it was extended to help the student learn how to handle new learning challenges.

4.4.2 Unmanned Aerial Vehicle (UAV)

The UAV domain was the first curriculum introduced into the BL program after Blocks World. The initial curriculum design and simulator were provided by Sarnoff, Inc. The domain focused on *UAV control*, and its concepts were separated into three Units:

- Unit 1 taught concepts central to operating the UAV, such as checking fuel levels prior to takeoff, flying from one waypoint to another, raising and lowering the landing gear, etc.
- Unit 2 taught concepts central to operating the UAV camera.
- Unit 3[5] taught concepts central to recognizing (simple) scenarios on the ground, ranging from recognizing low-level features such as relative distance (e.g., “near”) to more abstract features such as the conditions under which truck movement might be considered suspicious.

The difficulty of the concepts in the UAV domain ranged from straightforward (e.g., teaching a procedure as a linear sequence of steps) to highly difficult (e.g., ambiguous examples) and so helped focus the program on a “sweet spot” in Phase 2, in terms of what sorts of problems were reasonable for the e-student to handle. Some kinds of lessons – for example, those that explicitly taught syntax (class-hierarchy location, parameters, and type-constraints on parameters) – were eventually dropped as “too easy,” while others – for example, those that used examples to teach cause-effect relationships when multiple simulator actions could cause the same effect – were dropped as “too hard.”

The UAV curriculum initial design and simulator were provided by Sarnoff, Inc.

4.4.3 Armored Task Force (ATF)

The ATF domain curriculum and simulator were provided by Teknowledge. The curriculum was initially designed to teach concepts about force movement. Because of the similarities between force movement and control in the UAV domain, the program manager requested a change in focus, and the ATF curriculum was redesigned to focus on planning. The over-arching task of the redesigned curriculum was to teach the student an algorithm for “grading” a planned traversal by a company over an area with several different terrain features, based on the time it would take for a company to follow the route segments that comprised the overall path. Factors affecting time included not only the length of the route segments, but the terrain quality for each segment, the capacity of the company to handle certain kinds of terrain (specifically, mine fields, which could be crossed if the company was equipped with a mine plow), and how frequently the company was required to change formation. As an additional learning challenge, the plans did not explicitly specify formations, so the curriculum also taught a procedure for filling in this information, which the student had to supply in order to grade a planned traversal correctly.

[5] What we call “Unit 3” here is actually referred to in the UAV Curriculum Manuals and Design Document as “Unit 4” – the “original” Unit 3 was unimplemented, so that Unit 4 is the third of three implemented UAV units.

The ATF curriculum units included:

- Unit 1 taught the syntax for the `SetVelocity` function, a concept from the original ATF design that had been implemented prior to the redesign.
- Unit 2 taught the `CompanyHasMinePlow` predicate. Like the `SetVelocity` function, this concept originated from the original design; however, this predicate was used later in the curriculum to help assess the time it would take to cross terrain that contained mines.
- Unit 3 taught all of the concepts that emerged in the redesign – all of the predicates, functions, and procedures needed to grade a plan for company traversal.

4.4.4 International Space Station (ISS) (Version I)

The ISS domain and simulator were provided by the AI research division of Stottler Henke [11]. The general theme of this curriculum was diagnosis. The e-student was placed in a simulated environment where it would receive alerts, and the instructor would teach it the significance of those alerts, a method for generating hypotheses about what problems those alerts might signify, and a procedure for ranking the hypotheses in terms of what should be "explored" first. The central metaphor used in the curriculum was a "whiteboard" – an artifact both shared and independently modifiable by the teacher and the student – that was used to maintain an evolving model of the space of hypotheses as to what might be wrong with the space station. Diagnosis in this curriculum thus consisted of modifying the whiteboard in a way that allowed hypotheses to be expanded, and priorities set to make the expansion sensible. The difficulty level for learning this diagnostic procedure was deemed sufficiently high that the teaching of repair methods was deferred for later phases (see following section on ISS version II). In its formalized, deployed form, the curriculum consisted of eighteen concepts across two units:

- Unit 1 taught the notion of an abnormality, and the various conditions under which a notification from the ISS system might indicate an abnormality. Here, too, the notion of a whiteboard was used, and the student was taught how to register abnormalities on the whiteboard.
- Unit 2[6] taught the various functions and procedures needed to navigate and modify the diagnostic hypothesis space via the shared whiteboard. This involved teaching the student how to "reason backwards" from a hypothesized event that could explain an abnormality to the various types of events that could cause that event, as well as how to "look ahead" to possible longer-term effects (such as mission failure or loss of the space station) if the ultimate cause (revealed by the reasoning-backwards process) were to go unaddressed.

This curriculum was highly ambitious from both an instructional and software engineering point of view. The hypothesis space, modeled via the whiteboard object, was capable of growing to a size that made updates to it non-trivial in terms of memory management in the IL virtual machine. Moreover, the procedures for analyzing the hypothesis space were difficult to describe in a "natural" fashion, as they required a precise understanding of the formal structure of the

[6] Again, there is a numbering issue with this curriculum. As originally designed, the ISS curriculum consisted of a large number of concepts to be taught in eight units. Due to the complexities involved and limitations with the ISS simulator, only two units were implemented.

whiteboard, its constituent hypotheses, and the relationships among hypotheses. As a result, learning algorithm writers had trouble understanding the general nature of the learning problem, making learning algorithm debugging hard.

A major source of the complexity was that the ISS simulator was unable to support the whiteboard model, which ultimately had to be represented in InterLingua as a modifiable object that could be passed back and forth. A conscious decision for Phase 3 was to update the ISS domain with a new, more robust simulator, and to focus the domain away from the whiteboard metaphor, and towards a simpler model of diagnosis and repair. The result was version II of the ISS domain, which was developed as a completely distinct curriculum from ISS I.

4.4.5 International Space Station (version II) (ISS-II)

The ISS II domain and simulator were also provided by Stottler Henke Associates. The curriculum featured seventeen concepts across 3 Units:

- Unit 1 taught concepts central to alert-monitoring, such as the notion that an indicator has changed status, or that a new, unacknowledged alert has appeared.
- Unit 2 consisted to two independent rungs, developed prior to the finalization of the ISS II design, that taught the student to recognize that changes to an indicator status mean that a control flow valve needs to be adjusted, and a procedure for identifying which valve(s) needs to be adjusted.
- Unit 3 taught diagnostic predicate/repair procedure pairs needed to respond effectively to the indicator status change events introduced in Unit 1.

Version II of ISS was significant not only because it fulfilled the early promise of ISS I by combining diagnosis with repair, but also because it supported two Phase 3-specific learning challenges: reactivity (i.e., learning to react in response to stimuli from the simulated environment), and reasoning about change over time. Until Phase 3, the IL language was restricted in its ability to handle these kinds of challenges, and the introduction of the new ISS simulator provided the program with an opportunity to explore them. Indeed, the Phase 3 Hidden Domain curriculum was designed so as to contain these challenges, and of the diversity domain simulators, only the ISS II simulator was in a position to support them.

4.5 Hidden Domain (and Simulators)

A common problem with learning research and technology is overfitting of learned results to training and test data. For this reason it was considered highly desirable to show that the learning capabilities claimed by the learning teams were demonstrable in at least one problem domain of which the teams themselves had no prior knowledge: the so-called *hidden domain* (henceforth, the HD) of BL evaluation.

Whether or not a certain representational capability is exhibited by a finite convergent learning behavior is a function of the dispositional state to which the learning system converges as the result of training: which amounts to saying that the test of whether a system has learned to satisfy one condition *O* if another condition *I* holds is whether the system will produce *O* in *any* situation where *I* is satisfied. The practical problem of establishing this general result on the basis of a finite number of test cases faces the challenge that, for any finite test set, the logical possibility exists that the learner has in fact learned a condition other than *I* that happens to be common to all of the tests. One way of obtaining defensible assurance of generality is to provide the learning system with test cases which the system designers could not have anticipated and where the input condition holds, the persistence of convergence under unanticipated conditions serving as a strong indicator that adaptation is not overfitted to the training domains.

4.5.1 Development Process

As was the case with the diversity domains, the hidden domain supported an electronic development environment for algorithm learning, and as with the diversity domains, this environment presupposed interaction with a domain-specific simulator governed by a curriculum specification comprised of concept-specific rungs. Multiple lessons distinguished by natural instruction method or NIM were targeted to each rung. The natural instruction methods used were identical in format to those employed by the diversity domains and were agreed on by a consensus process between learning and evaluation teams, resulting in formally specified NIM contracts. The domain choice, informal domain design, electronic domain simulator, and Application Program Interface (API) were produced by the domain provider (Stottler Henke Associates) in consultation with the curriculum provider (Cycorp). The curriculum, consisting of formally specified electronic lessons teaching a partial order of concept rungs, was designed and implemented by Cycorp personnel.

4.5.2 Content and Curriculum

The operational learning environment selected for the HD concerned misconfigurations and component failures in satellite ground station equipment [12, 13], with training scenarios originating with actual experiences with ground stations for scientific satellites provided by a Stottler Henke Associates subject matter expert, Dr. Manfred Bester. The original HD curriculum consisted of five units and 23 rungs. Unit 1 was comprised of two rungs: how to identify abnormal indicators for a satellite tracking pass, and how to list components showing abnormal indicators. Unit 2 concerned the classes of equipment malfunction which were sufficient conditions for a component showing abnormal indicators. There were six such classes, corresponding to six IL predicates, with a single predicate taught in each unit rung. Unit 3 concerned repair procedures for each of the taught fault conditions: there were six such repairs (corresponding to the six failure modes of Unit 2), and six Unit 3 rungs, with one rung corresponding to each repair procedure. Unit 4 taught conditionalizations linking failure modes and repairs: i.e., that each failure mode was in turn a sufficient condition for carrying out the corresponding repair procedure. Finally, Unit 5 taught a global diagnostic regimen for monitoring for abnormalities and applying repair procedures as needed, that would serve in all of the Stottler Henke fault scenarios. It consisted of three rungs. First, a repair procedure was taught that sequentially tried fixes on uninitialized, misconfigured, and inappropriately offline components, followed by fixes of intrack error, elevation error, and azimuth error, if the component on which the fix was being attempted was the tracking antenna. Second a secondary repair procedure was taught that took a list of components as input and, for each component element in the list, found the immediate predecessor in the tracking station dependency graph
and executed the previously taught Unit 5, rung 1 repair procedure on that predecessor. Finally, a comprehensive diagnose-and-repair script was taught, that began with an assemblage of a list of all abnormal components, followed by an execution of the basic repair procedure on every element of the list, followed by regeneration of the list of abnormal components, followed by execution of the secondary repair procedure on the elements of the secondary list.

This was the extent of the Phase 2 curriculum. Its seemingly over-determined character was in fact necessitated by the initial semantic limits of IL; in particular, there was no provision in the language or the curriculum framework for a student to take any action not explicitly prescribed by the teacher, which made it impossible for a student to act in any way that had not been taught to it as a named procedure, thereby ruling out 'emergent' behaviors that came about as a result of monitoring for salient conditions. In Phase 3, IL semantics were extended in ways that allowed for precisely this functionality, and the HD curriculum was broadened in order to take account of the new capabilities. A new unit, Unit 3A, of six rungs was added wherein the student was taught success criteria for the six repair procedures taught in Unit 3, and the diagnose-and-repair procedure of Unit 5 was refined so that success criteria could be used to gate additional repairs in a way that reduced performance of redundant or unnecessary repair procedures. Also, another new unit, Unit 6, was added to teach trigger conditions for the diagnose-and-repair procedure. It consisted in five rungs: the first taught truth conditions for when the satellite tracking task schedule had changed; the second and third taught predicates for monitoring alert indicators that, if learned, jointly enabled the student to conclude that a readout was alerting as abnormal; the fourth related these predicates to trigger conditions for initiating the diagnose and repair procedure, and the fifth taught a reactive procedure for applying the modified diagnose-and- repair procedure of the new Unit 5, relative to these trigger conditions.

4.5.3 Phase 3 NIM Relaxation Trajectories

In addition to introducing elements that implemented novel learning challenges, the Phase 3 curriculum also modified lessons via strategic ablations of NIM protocols. Referred to as Relaxation Trajectories (RTs), several were implemented for the HD in Phase 3, notably removal of concept-identifying control utterances, removal of argument-signature-revealing utterances, removal of explicit telling of sufficient conditions when teaching if/then/else clauses, and generalization of utterances stating relevant conditions.

4.5.4 Hidden Domain – Diversity Domains Comparison

A natural concern in testing system performance scaling is whether the scaling test or tests actually belong in the operational range of the functionality being tested. Thus, it was deemed important to show that the HD curriculum was in balance no harder than the diversity domain (DD) curricula, and that the totality of the learning tasks of the diversity domains was such that a student that performed adequately on all of them could be expected to deliver a comparable performance on the HD. For this reason, a detailed comparison of the HD and DD curriculum frameworks was undertaken. The conclusion was that the coding syntax of the HD framework was at least no more complex than the syntax of the most complex DD curriculum lessons, and that the induction task required in each HD lesson was homomorphic with respect to some induction task in the diversity domains, when the propositional inputs and desired outcomes of the tasks were rendered in the IL ontology.

4.6 BL-ISR

In Phase 3, we applied BL methodologies to the ISR domain. This domain provides a good testing ground for BL techniques for a number of reasons. Making the most effective use of ISR data requires that it be combined with information of diverse sorts – background information on patterns of life, geospatial entity data, human intelligence (HUMINT), signal intelligence (SIGINT), etc. In addition, analysts discover relevant patterns in the field, making it difficult if not impossible to pre-program automated search for such patterns. There is therefore a strong need for a system that allows the analysts themselves to teach an automated fusion system new patterns and rules. The tools developed in BL provide a natural means for accomplishing this sort of instruction. The following subsections describe sample scenarios involving the analysis of ISR data, the representation of such scenarios in our system, and the instructional interface we developed for teaching analytic rules and procedures for such scenarios.

4.6.1 ISR Analysis Background

ISR activities involve the collection of data in support of military or national security objectives. Data are collected through sensors on satellites, manned and unmanned aerial vehicles, or ground-based sensors. The data collected comes in a variety of forms, including optical, radar, electronic signals, and infra-red. For our purposes, a type of radar data known as GMTI data is important. This data is generated by a radar system that detects targets moving on the ground (e.g. vehicles).

This data can be used for a variety of purposes. To be useful, the individual detections need to be stitched together to form *tracks*, temporally ordered sequences of detections inferred to be from the same target. Track data can be enriched with additional layers of information such as:

- Road networks
- Locations of interest (e.g. houses, factories)
- Communication events (e.g. cell phone calls)
- Threat events (e.g. Improvised Explosive Device (IED) explosions)
- Pattern of life analysis
- HUMINT

Data that has been processed and enriched in this way is now at the symbolic level, not the pixel level, and BL techniques can be applied to data at this symbolic level to solve learning problems of interest to analysts. The additional, non-track, sources of information supply a context for interpreting track data. For example, a pattern of life analysis tells the analyst what activities in a region are normal – e.g. patterns of traffic to and from a workplace or a mosque, times at which farmers plow their fields, transport of livestock across a border, and so on. Patterns that deviate from the normal patterns of activity will be of interest to analysts and merit further investigation. Similarly, the recent occurrence of certain events such as IED explosions will heighten interest in track patterns that might be interpreted as indicators of an impending event (e.g. locals avoiding routes that normally have a lot of traffic).

4.6.2 ISR Scenarios

We investigated a number of scenarios of interest to ISR analysts. Given our available ISR simulation data, we were able to represent three scenarios of interest:

- Suspicious Meeting
    - Multiple tracks converge on a remote location that rarely sees traffic
    - Tracks arrive at location close in time to one another
    - One of the tracks starts near a known safe house
- Clandestine Theft Scenario
    - Track ends in deserted location at unusual time (3 AM)
    - Location is near facility storing materials usable by insurgents

- Impending event (social knowledge)
    - Area with normally heavy traffic during a certain time of day (e.g. morning rush hour) has significantly decreased traffic
    - Knowledge of an impending event (e.g. an IED explosion) may have spread through the local populace

For the first two scenarios, we developed instructional scripts for teaching the concepts involved in those scenarios. The instructional sequences for the suspicious meeting and clandestine theft scenarios are described in detail in ISR Jr User's Manual [14].

4.6.3 System Architecture

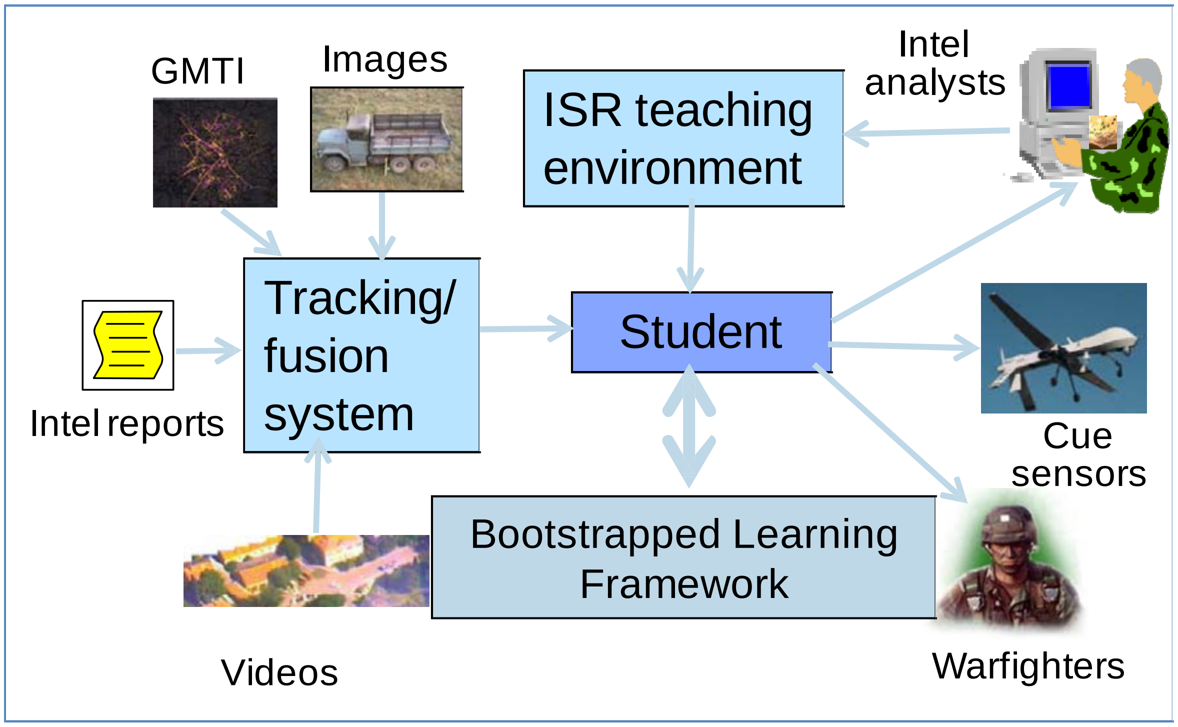


Figure 3: The BL-ISR Architecture

Figure 3 shows a high level view of the BL ISR architecture. The patterns of interest are taught to the student by an analyst, with the BL Framework being used as a messaging and integration system for the student and teaching components. Data from multiple heterogeneous sources is fed into a tracking/fusion system that produces higher-level representations in symbolic form used by the e-student in inferring instances of the learned patterns. The analyst will also teach priorities for different patterns and depending on the priorities of inferred pattern instances, the e-student will issue requests to cue sensors for additional information or will issue alerts to interested parties.

4.6.4 Analysis and Instruction Stages

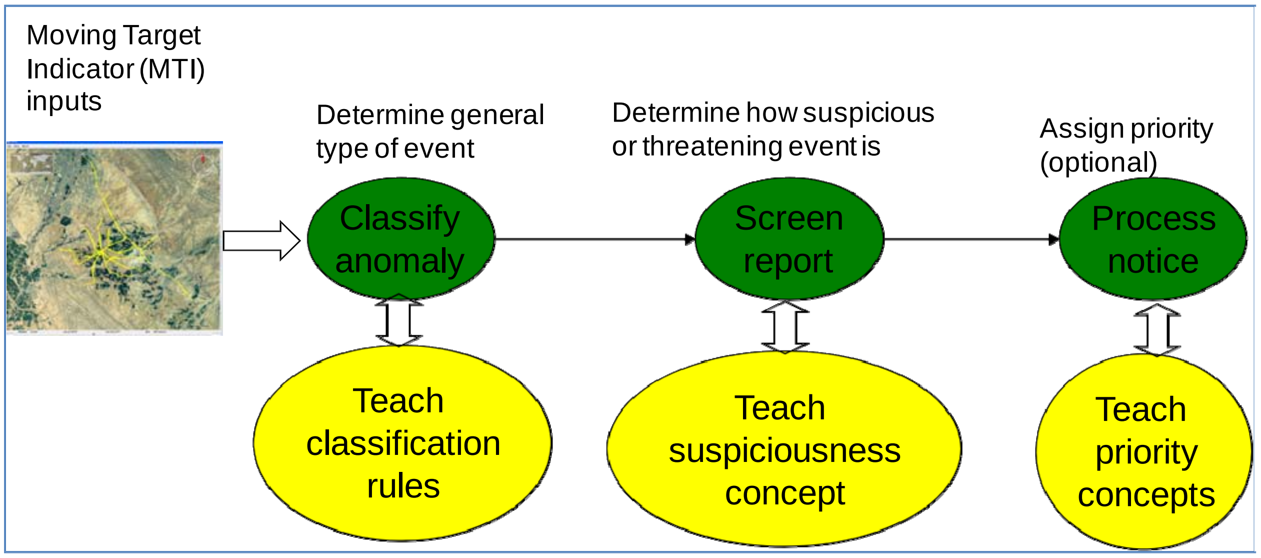

Moving Target Indicator (MTI) inputs
Determine general type of event
Determine how suspicious or threatening event is
Assign priority (optional)
Classify anomaly
Screen report
Process notice
Teach classification rules
Teach suspiciousness concept
Teach priority concepts

Figure 4: The Three Stages in BL-ISR Analysis and Instruction

We broke the analysis process up into three stages, with corresponding types of instruction for each stage. We assume analysis starts with the detection of some anomaly (or more generally, some pattern of behavior that the analyst believes merits further attention). The first stage of analysis is to classify the anomaly – to determine what kind of event is happening. It might be a number of vehicles converging on a particular location or a vehicle stopped in a location that is normally deserted. After classifying the anomaly, the next analytic task is to determine whether there is cause for concern – is the event an indicator of a worrying problem or issue – is something suspicious happening that may pose a threat? Finally, the analyst must prioritize the alerts in order of their importance. In the BL ISR effort, we investigated different techniques for performing the first and second of these analysis tasks – classifying and determining suspiciousness.

4.6.5 Instruction Techniques

A number of instruction techniques were explored to support BL-ISR. These include:

- *Natural language instruction*: giving a rule in natural language, with feedback about the correctness of the generated formalization. This is a version of *Learning by Telling*.
- *Learning by Example*: apply Inductive Logic Programming to learn from a set of labeled examples. There is a continuum of teaching techniques between Learning by Telling and (pure) Learning by Example since Learning by Example allows the instructor to suggest starting rules. The starting rule could be a trivial one ("Classify every track convergence as suspicious") that gets refined by examples; or the starting rule could be close to the final, correct rule, in which case a much smaller set of examples is needed to refine the rule.
- *Templated Learning of Procedures*: This allows the instructor to select, specialize, and combine procedures from a library of procedure templates.
- *Learning of Utility Functions:* In this form of learning, patterns of behavior, such as vehicular movement inferred from GMTI data, are explained in terms of utilities of agents (agents prefer this route to get from A to B over this other route). Given the

learned utility functions, anomalies are detected when they cannot be explained by the utility functions (this agent is taking a route from A to B not predicted by the utility functions for paths).

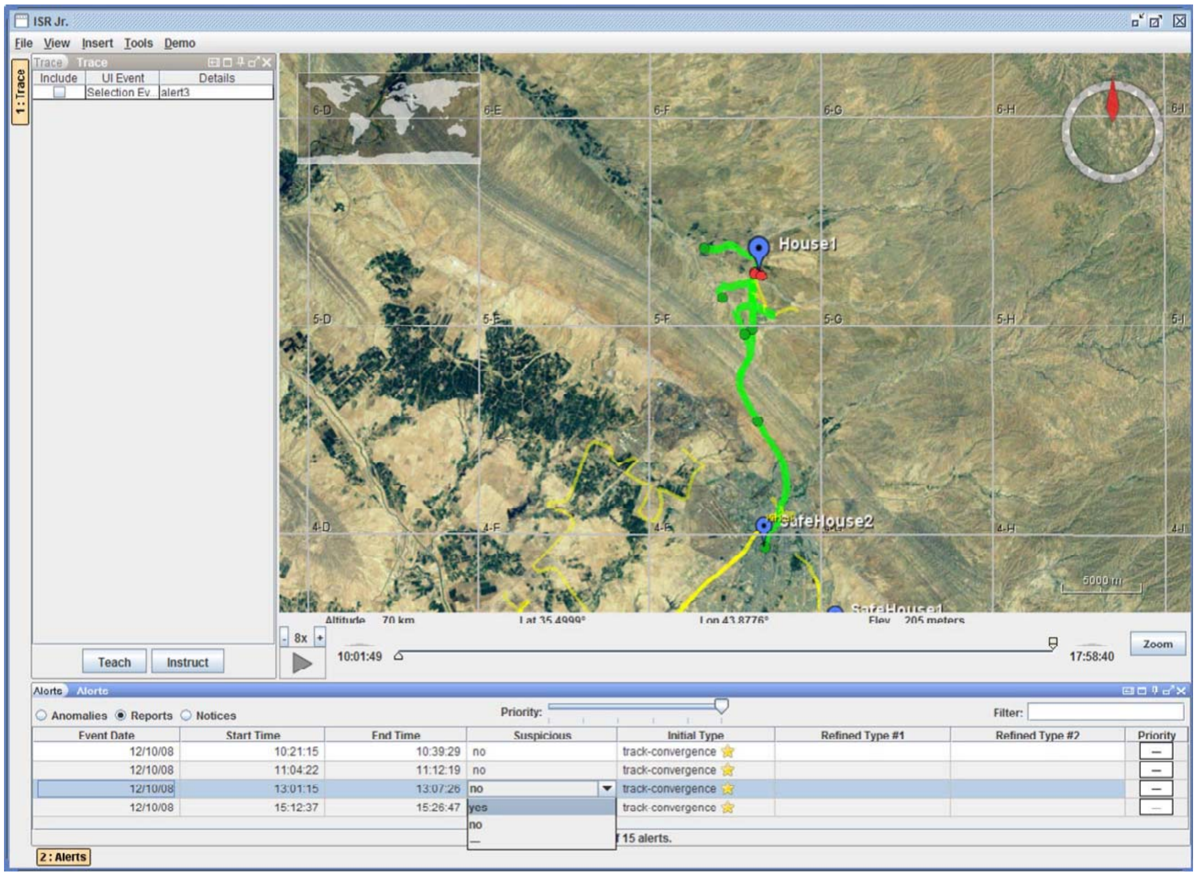
An overarching theme of all these instruction techniques within the BL ISR framework is that they are all *situated*. The meaning of situated instruction is spelled out in the next section.

4.6.6 Situated Instruction

A common way of extracting knowledge from experts is by using knowledge engineers as intermediaries to translate subject matter experts' knowledge into a formalism usable by machines. There are many pitfalls in this approach: Subject Matter Experts (SMEs) have difficulty articulating their knowledge in a format that knowledge engineers can grasp, key assumptions obvious to the experts may not be made explicit, and the models/representations created by the knowledge engineers are difficult for the SMEs to evaluate. Another problem is that the patterns that need to be represented might change over time or vary from place to place, so that the knowledge that needs to be represented is highly localized and not easily captured outside of the specific context in which it is applied. For these reasons, much research on knowledge acquisition has focused in the last decade on making it easier for the SMEs themselves to enter knowledge into an inference system. This has taken the form of interfaces for natural language, or structured natural language; various types of graph-based or diagrammatic representations; form-based interfaces; and libraries of knowledge components. These techniques have improved the knowledge acquisition task to some degree, but they have limitations, such as:

- Diagrammatic representations can quickly become unwieldy and tedious to construct
- Natural language interfaces can be helpful but are often not sufficient on their own
- Controlled natural language interfaces present some of the same problems as formal representations (user does not understand intended syntax or semantics, or makes incorrect assumptions about them)

A more basic problem with these approaches is that SMEs knowledge often cannot be easily articulated. Research on the learning of cognitive skills has shown that "as someone becomes an expert, knowledge often becomes encoded as perceptual-motor skills rather than rules and therefore less accessible to conscious formulation" [15]. Quite frequently, the problem is not the lack of a suitable representation language, but rather the SME's knowledge being inaccessible to conscious formulation (because it is embedded in perceptual or motor skills exercised in a particular environment).



Include	UI Event	Details
☐	Selection Ev.	alert3

Event Date	Start Time	End Time	Suspicious	Initial Type	Refined Type #1	Refined Type #2	Priority
12/10/08	10:21:15	10:39:29	no	track-convergence			—
12/10/08	11:04:22	11:12:19	no	track-convergence			—
12/10/08	13:01:15	13:07:26	no	track-convergence			—
12/10/08	15:12:37	15:26:47	yes	track-convergence			—

Figure 5: Screenshot of the BL-ISR Instructional Interface

To overcome this problem, the BL ISR instruction framework uses an instructional interface that mimics the analyst's work environment and ties instruction to the actions the analyst performs in doing his or her work. Figure 5 shows the instructional interface used in BL ISR. There is a map display showing tracks, structures, and other geographic features. The bottom panel displays alerts about events. A typical analytic task is to select a particular event and learn more about it, first by classifying it as a particular type of event and second by assessing whether there is anything suspicious about the event that warrants further investigation. The analyst can fill in this information in fields on the right-hand side of the Alerts panel. The interface also allows the analyst to switch to Instructional mode, in which she instructs an e-student in the rules used in making classifications and assessing suspiciousness (more detailed scenarios can be found in the ISR Jr User's manual). The instruction is situated in a concrete environment familiar to the analyst and elicits knowledge from the analyst in the context of the same actions the analyst normally performs in doing his or her analysis. The burden of articulating rules is eased by virtue of applying machine learning to concrete examples, while at the same time the number of training examples needed is minimized through the instructor's guidance toward the right rules.

4.7 Evaluation

We utilized several methods to evaluate different aspects of the BL program, including studies to explore approaches that humans take in teaching other humans as well as approaches they might take in teaching an e-student (Section 4.7.1), establishing human-performance baselines on the hidden domain (Section 4.7.2), and automated testing of the e-student on the hidden domain (Section 4.7.3).

4.7.1 Approaches to Instruction (Phase 1)

In Phase 1 we undertook two studies to explore approaches humans take for instructing others. The first explored human-human instruction; the second investigated human-e-student instruction. In this section we present a summary of our findings. Our published papers [16, 17, 18] and BAE Systems technical reports [19, 20, 21, 22] provide details of the protocols and results.

The focus of the first study was to compare human instruction of other humans to the instruction types specified by the NIM created for the Bootstrapping Learning Project. It examined three transcripts collected from other sources [23, 24, 25], capturing three types of interaction: human-human interaction, naïve human interaction with a human tutor disguised as a machine, and human tutoring with an interlocutor (not actually a student) present. We found examples of all NIMs within the corpora. In addition, we noted a number of social aspects to the interactions (e.g., hedging, humor, and exclamations of surprise) that would not necessarily be exhibited in human-e-student instruction. It remains an open question as to whether social considerations should be incorporated into a BL system.

The focus of the second study was to gain insight into how a human teacher would instruct an e-student [16]. Personnel from the University of Texas at Austin and Cycorp performed an initial case study using Blocks World as the domain, echoing the initial curriculum development for the e-student, and a Wizard of Oz (WOz) [26] methodology in which a human teacher's instruction were translated into IL for the e-student by a human translator. In addition to the quantitative data (e.g., methods used, teaching time, and e-student performance) they collected qualitative data about their experience, any difficulties they encountered, and their model of the e-student both before and after the teaching session. Five subjects participated, attempting to instruct the e-student in two tasks – how to construct a stack of three blocks, and how to build a simple doorway by placing a lintel across two stacks of blocks. Each subject was asked to assume the role of a human teacher, and was told that the e-student could be considered on the same level as that of a bright two year old. All five subjects succeeded in their instruction. Table 1 summarizes our observations of the experiment.

Table 1: WOz Observations for a Human Teacher Instructing an E-Student

Observation	Impact
All of the human teachers ended up using a bottom-up approach to teaching (possibly due to capabilities of the e-student). Some human teachers initially used a top-down approach but became frustrated and reverted to a bottom up approach.	All BL curricula to date have been authored using a bottom-up structure. In the Phase II and Phase III evaluations with human learners (rather than e-students), we have found that human subjects often prefer a top-down instructional method. It is an open question as to how an e-student might learn with a different lesson structure.
All of the human teachers overestimated what the e-student knew and could do, assuming knowledge of primitives such as "choose a block" or "look for a clear	We believe the domain-independent e-student needs a minimal level of injected knowledge to support "basic human competencies". This can be achieved through the use of background knowledge which can

space."	be injected or built into an e-student.
Many of the human teachers employed repetition and mnemonics when teaching.	Instructional interfaces should support natural human methods for teaching, including support for informalities. The Curriculum Team is currently exploring this issue.
The human teachers differed in their assumptions regarding the linguistic capabilities of the e-student.	Instructional interfaces should mask the linguistic limitations of an e-student and/or a teacher should have a way to query an e-students capabilities and understanding.

4.7.2 Human Benchmark Experiments (Phases 2 and 3)

In addition to concerns about whether the HD presented an adequate test of scaling scope, concerns also arose in Phase 2 regarding whether the learning requirements of the BL program were in fact within the range of capabilities of a competent human adult. In order to address these concerns, personnel at BAE Systems, Cycorp and the University of Texas at Austin collaborated on adapting the HD curriculum to a multimedia training regimen for human students. Study of the outcome of this training showed that human competences with respect to the adapted human curriculum were sufficient for the tasks, and in some respects, far in advance of the abilities exhibited by any of the automated learners (for example, the human learners propensity for converging to appropriate conditional inductions made it possible to entirely dispense with Unit 4 in the human curriculum). Programmatic interest in the strengths and weaknesses of human learners as compared with machine learners justified continuing human subject testing and study throughout Phase 3. We published the results of these experiments [16, 18]; the detailed reports are in BAE Systems technical reports [19, 20, 21, 22]. In this section we provide highlights of our findings.

Our primary results established the passing threshold requirement for e-student performance. In Phase 2, 28 human subjects participated, establishing a baseline performance requirement of 91%. In Phase 3, 19 human subjects were tested on the full curriculum, establishing a baseline performance requirement of 81%. An additional 40 subjects were tested on subsets of the curriculum to investigate NIM-effectiveness. In addition to these baseline requirements for the e-student, we identified a number of lessons learned in developing the protocols to conduct our experiments and fairly compare human and e-student performance. Human and e-students differ in fundamental ways that make it difficult to create analogous contexts without providing one side with undue advantages over the other.

First, e-students have perfect memory of all lesson material they have seen. We compensated for this in human testing by allowing subjects to take notes and review lessons if desired. Human students also have a harder time interpreting formal language or concepts expressed in other "unnatural" ways. Because of this, we were forced to produce a more natural version of the e-student curriculum for the human students, introducing possible confounding factors into the comparison. On the other hand, human students have a greater understanding of the semantics of words and have the ability to gain domain knowledge outside the formal channel of the curriculum, such as through voice intonation or gestures inadvertently expressed by a human teacher. We addressed the issue of leaky semantics by being careful that our choice of terms didn't leak unintentional knowledge, and also by going through several preliminary iterations of the curriculum. Interestingly, increased semantic understanding was also occasionally detrimental to human subjects, when the knowledge leaked by terms was misleading.

4.7.3 BL Student Performance (Phases 2 and 3)

For the e-student, the performance on the final exam in Phase II was 100%. In Phase III, the e-student was able to complete all of the tasks correctly. However, when a penalty is introduced for rungs that had to be injected, rather than learned, the performance is reduced to 94% (i.e., (rungs – injections) / rungs). That is, for 5 of the 7 final exam problems there was one rung where the e-student was unable to learn the concept and was supplied with the concept description (injected) so the e-student was able to continue onto other rungs. It is important to note that in Phase III the student was only tested with the relaxed formalisms and the additional Unit 6 on trigger conditions. The e-student was not tested on the updated repair procedures in Unit 3 or monitoring the time bias in Unit 7. Additionally, in both phases of testing we collected several informal testing results, including intermediate rung tests, individual NIM-effectiveness, and diversity domain testing results [27, 28, 29, 30, 31, 32].

4.8 Seedling Efforts

In this section we provide an overview of the findings from two seedling efforts: Framework Adoption (Section 4.8.1) and PROWL (Section 4.8.2). Additional details can be found in BAE Systems technical reports [33, 34].

4.8.1 Framework Adoption

The Framework Adoption seedling effort resulted in a survey of recent research related to framework adoption, the development of a prototype Eclipse-based Integrated Development Environment (IDE) plug-in, the development of an example system with the widely used Hibernate persistence framework along with a scenario for demonstration, and a summary report [33]. The survey indicated that there is much new research in the topic of mining source code repositories for patterns that has yet to be harnessed for the purpose of distributing knowledge and integration into the user experience. The prototype IDE plug-in successfully demonstrated real-time monitoring of user programming focus, recognition of patterns from a pattern repository, and recommendation of the most relevant pattern. The example system, along with the tasks for a demonstration scenario and experimental protocol, outline an exploratory investigation of how users would interact with a framework adoption system.

4.8.2 PROWL

Under the first thrust, we investigated PROWL concepts using Army Knowledge Online (AKO). In retrospect, AKO may not have been the optimal choice for a vehicle to demonstrate the PROWL concepts. The AKO system has received substantial criticism with regards to its speed, various areas of functionality, complex security requirements, effectiveness, and compatibility with web browsers, particularly from its daily users. The PROWL team also encountered significant resistance from users to supporting our experiments, probably due to the ongoing performance problems and dissatisfaction with the existing user population with AKO in general. Nevertheless, the concepts described and investigated under this seedling have demonstrated the promise of this approach applied to large, enterprise text corpora.

Under Thrust 2 we explored three broad areas associated with crowd-based activity grounded in a military context:

- Shared Sensing – patrollers can see location of other team members and any points of interest or adversarial agents they mark up in the environment
- Coordinated Action – patrollers act in concert with others to achieve a goal, and are able to rapidly and continually replan in response to the changing conditions
- Collective Reasoning – patrollers and their leaders respond to strategic and tactical goals and collectively analyze new information by marking up the virtual space in real time

We developed seven uses cases, including one based on a real-world example. We also identified two supporting technologies, available today, that would support and enable a PROWL-like system to be developed and deployed [34].

5.0 Conclusions and Recommendations

The BL electronic student brings virtually no domain knowledge to its learning tasks. The only knowledge built-in to the system (i.e., the only place where it depends on the actual "spelling" of terms used in the ITL) is with respect to a limited number of control keywords in the ITL (e.g., "utterance" or "imperative"). To repeat, no domain knowledge is built in to the learner, it must either be taught or provided as background input knowledge. Because of this, the electronic learner can be viewed as applying weak methods to solve a domain independent learning problem. This view raises interesting questions about the adequacy of information that can be conveyed via syntactic structure (i.e., the electronic learner would have learned the exact same concepts had we uniformly encoded terms in the curriculum that mean something to humans and replaced them with syntactic gobbledygook).

Both the human subject testing and DD-HD comparison efforts of BL raise larger questions about the semantic basis for comparing curriculum frameworks. In comparing two curricula, two rungs, or two lessons, a fundamental question that must be asked is whether such similarity or degree of complexity as is identified is relevant to the cognitive architecture of the student. Where complexity is measured in syntactic terms (for example, nesting depth of logical operators), it is at least possible that a student's internal representation might be such that external expressions with distinct syntactic complexity would map onto internally isomorphic student representations: depending on the student's processing requirements (and of course,
upon the intrinsic cost associated with the mapping), the student might therefore be indifferent to overt differences in external complexity. And where information is measured probabilistically, utilizing metrics explicated by Claude Shannon, it is worth reminding ourselves that these probabilities are calculated with reference to a possibility-space that reflects some agent's model of what features are relevant and what world states are and are not possible: 'information value' is still computed relative to the world model of the agent. Although our curriculum comparison efforts assumed that the SRI student's learning biases essentially approximated those of a suitably skilled human, this is really an open question amenable to further study. Also, one of the intriguing indications to emerge from the Phase 2 analysis and subsequent DD and HD testing is that there may be complementary domains of complexity such that simplification in one area is possible only through incurring some representational cost in some other, so that hidden expenses may attend many, and perhaps most methodologies for simplifying definitions – e.g., eliminating step-wise sequences inside of while loops to accommodate known limitations of learning modules may come at a price of introducing a higher 'nesting factor' for the target definition overall, which might in turn pose difficulties for *other* learning modules. Phases 2 and 3 provided us with almost no empirical data regarding questions of how or indeed, whether, such representational changes impacted performance in the MABLE system, so this would therefore seem to be a rich field for future scientific investigation.

Finally, a fundamental feature of the IL framework is that the same world model to which the student has access as background knowledge is also accessed by the teacher, and to implement the actions of the world agent. Among other things, this helps to insure the percepts that the world agent produces are structured in a way that demonstrably conforms with both the ITL explanations generated by the teacher and the target IL definitions that the student is expected to compile as the result of the curriculum. It is obvious, however, that the ubiquity of this framework could not be maintained in a scenario wherein the electronic teacher was replaced with a human trainer, just as scaling to real-world applications would entail replacing the IL world agent with the real world. In such circumstances, even the relative guarantee that the teacher and student share a common representational framework would be lost (even if the teacher were to use a controlled vocabulary, there would no longer exist a canonical template in the form of a common world model against which to test this vocabulary to insure that it is deterministically translatable into a defCode implementation). In addition, the real-world extension faces the added challenge of insuring that the structures in the actual world that the student and teacher manipulate generate a percept stream that is intelligible for both the teacher and the student. Such interoperability issues constitute a critical research topic for automated learning.

Even with the above caveats, the BL program did demonstrate perhaps surprisingly good and general results on both the Phase 2 and Phase 3 hidden domains. These results lend credibility to the instructional approach to learning. In addition to the limitations mentioned above there also remains the issue of how a learner can produce a result that is deployable in a useful operational context. In the BL program the learner simply produced an IL program that was executable in a suitable context (e.g., to compute a result, answer a test question, or execute a procedure in the simulated world). In the real world a learning system would have to deploy its results in some specifically suitable context (e.g., as an executable plugin in some larger performance system, as a command sequence to some robotic device, or perhaps as a set of instructions to be performed by a human co-agent). The domain independent success and the above limitations point to an opportunity to apply the Bootstrapped Learning technologies in a single, specific learning environment. Rather than strive for the breadth of applicability as in the original program, focusing on learning and performance in a specific domain (e.g., ISR analysis) would provide the opportunity to overcome the limitations discussed above, stretch the bootstrapping approach to an extended training regimen (extended over both time and conceptual coverage), and provide the opportunity to assess success in a concrete domain with specific success criteria.

BIBLIOGRAPHY

1. Oblinger, D., 2006, “Bootstrapped Learning: Creating the Electronic Student that Learns from Natural Instruction,” AAAI Briefing, http://www.darpa.mil/ipto/programs/bl/docs/AAAI_Briefing.pdf.
2. Mailler, R., Bryce, D., Shen, J., and O’Reilly, C., 2009, “MABLE: A Framework for Learning from Natural Instruction,” In Proceedings of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AA- MAS 2009), Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10– 15, 2009, Budapest, Hungary.
3. Morrison, C., Bryce, D., Fasel, I., and Rebguns, A., 2009, “Augmenting Instructable Computing with Planning Technology.” In ICAPS ’09 Workshop on the International Competition for Knowledge Engineering in Planning and Scheduling.
4. Natarajan, S., Kunapuli, G., Page, D., Walker, T., O’Reilly, C., and Shavlik, J., 2010, “Learning from human teachers: Issues and challenges in bootstrap learning,” In AAMAS 2010 Workshop on Agents Learning Interactively from Human Teachers, www.ifaamas.org.
5. Curtis, J., 2009, “Bootstrapped Learning Interaction Language,” BAE Systems Technical Report TR-2231, December 2009.
6. Curtis, J., 2008, “Bootstrapped Learning Interlingua Transparent ‘Starter’ Languages,” BAE Systems Technical Report TR-2239, June 2008.
7. Blaylock, N., Curtis, J., Kahlert, R., Shepard, B., 2007, “Bootstrapped Learning Interlingua Parsing and Printing,” BAE Systems Technical Report TR-2178, 2007.
8. Curtis, J., Reubenstein, H., 2009, “Bootstrapped Learning Curriculum Language,” BAE Systems Technical Report, April 2009.
9. BAE Systems, 2012, “Bootstrapped Learning: Natural Instruction Method (NIM) Documentation,” BAE Systems Technical Report TR-2766, January 2012.
10. BAE Systems, 2012, “Bootstrapped Learning: Domain Documentation,” BAE Systems Technical Report TR-2767, January 2012.
11. Ludwig, J., Mohammed, J., and Ong, J., 2010, “Developing an international space station curriculum for the bootstrapped learning program,” In Proceedings of the March, 2010 IEEE Aerospace Conference, Big Sky, MT.
12. Mohammed, J., Davis, A., Ludwig, J., 2009, “System Design for Hidden Domain Curriculum and Simulator,” BAE Systems Technical Report, 2009.
13. Ludwig, J., Davis, A., Abrams, M., Curtis, J., 2011, “A Hidden Domain for Human and Electronic Students,” in 2011 IJCAI Workshop on Agents Learning Interactively from Human Teachers (ALIHT), July 2011.
14. Baxter, D., Frederiken, A., R., 2011, “ISR Jr. User Manual,” BAE Systems Technical Report TR-2750, October 2011.
15. Gluck, M., Mercado, E., Myers, C., “Learning and Memory: From Brain to Behavior,” Worth Publishers, 2007.
16. Berland, M., and Perry, D., 2009, “Novice Human Teachers of a Virtual Toddler: A Case Study,” Technical Report, The University of Texas at Austin. http://www.ece.utexas.edu/~perry/work/papers/090123-MB-blexp1.pdf
17. Grant, R., DeAngelis, D., Luu, D., Perry, D., Ryall, K., 2011, “Designing Human Benchmark Experiments for Testing Software Agents,” In Proceedings of the 14th International Conference on Evaluation and Assessment in Software Engineering (to appear), BCS eWIC.

18. Grant, R., DeAngelis, D., Luu, D., Perry, D., and Ryall, K., 2011, "Designing Human Benchmark Experiments for Testing Software Agents," In Proceedings of EASE April, 2011, Durham UK.
19. Perry, D., Sidner, C., 2008, "A Basic Plan for Human and Automated Student Comparison," BAE Technical Report TR-2224, November 2008.
20. Grant, R., DeAngelis, D., Luu, D., Perry, D., 2009, "Automated Student Human Benchmark Study: Phase II Report," BAE Systems Technical Report, 2009.
21. Perry, D., E., 2010, "A Plan for an Automated Student Benchmark Study Phase 3," BAE Systems Technical Report, June 2010.
22. Grant, R., DeAngelis, D., Luu, D., Perry, D., 2010, "Automated Student Human Benchmark Study: Phase III Report," BAE Systems Technical Report TR-2659, December 2010.
23. Cohen, P., 1984, "Pragmatics, Speaker Reference, and the Modality of Communication," Laboratory for Artificial Intelligence Research.
24. Johnson, L., 2008, "Chat Study, Virtual Factory Teaching System," Personal communication with Lewis Johnson, Allelo Corporation, 2008.
25. Einstein, J., 2008, "Gesture studies transcripts," Personal communication, MIT.
26. Dahlback, N., Jonsson, A., and Ahrenberg, L., 1993, "Wizard of Oz studies – Why and How. Knowledge-Based Systems," 6(4):258 – 266, 1993. ISSN 0950-7051. Doi: DOI:10.1016/0950-7051(93)90017-N. Special Issue: Intelligent User Interfaces.
27. Sidner, C., and Stromsten, S., 2008, "Evaluation Plan for an Electronic Student in Bootstrapped Learning," BAE Systems Technical Report TR-2172, April 2008.
28. Sidner, C., and Stromsten, S., 2008, "Evaluation of the Bootstrapped Learning Student Phase 1," BAE Systems Technical Report TR-2313, October 2008.
29. BAE Systems, 2009, "Phase 2 Evaluation Plan for an Electronic Student in Bootstrapped Learning," BAE Systems Technical Report TR-2531, November 2009.
30. Roberts, B., 2010, "Phase 2 Evaluation of the Bootstrapped Learning Student," BAE Systems Technical Report TR-2553, February 2010.
31. BAE Systems, 2010, "Phase 3 Evaluation Plan for an Electronic Student in Bootstrapped Learning," BAE Systems Technical Report TR-2631, September 2010.
32. Abrams, M., Curtis, J., 2011, "Phase 3 Evaluation of the Bootstrapped Learning Student," BAE Systems Technical Report TR-2711, July 2011.
33. Sullivan, G., 2009, "Framework Adoption Seedling Report," BAE Systems Technical Report TR-2768, January 2009.
34. Stephenson, T., 2011, "PROWL Seedling Final Report," BAE Systems Technical Report TR-2769, December 2011.

LIST OF ACRONYMS, ABBREVIATIONS, AND SYMBOLS

ACRONYM	DESCRIPTION
AFRL	Air Force Research Laboratory
AKO	Army Knowledge Online
API	Application Program Interface
ATF	Armored Task Force
BL	Bootstrapped Learning
BL-ISR	Bootstrapped Learning Intelligence, Surveillance, and Reconnaissance
BLADE	Bootstrapped Learning Analysis and Curriculum Development Environment
CDRL	Contract Data Requirements List
CL	Curriculum Language
DARPA	Defense Advanced Research Projects Agency
DD	Diversity Domain
GMTI	Ground Moving Target Indicator
HD	Hidden Domain
HUMINT	Human Intelligence
I2O	Information Innovation Office
IDE	Integrated Development Environment
IED	Improvised Explosive Device
IL	InterLingua
ISR	Intelligence, Surveillance, and Reconnaissance
ISS	International Space Station
ITL	InteracTion Language
Jr	Junior
MA	Massachusetts
MABLE	Modular Architecture for Bootstrapped Learning Experiments
ML	Machine Learning
NIM	Natural Instruction Method
PROWL	Probabilistic Relational Ontological Web Language
RDF	Resource Description Framework
ROL	Results of Learning
RT	Relaxation Trajectory
SIGINT	Signal Intelligence
SME	Subject Matter Expert
TR	Technical Report
UAV	Unmanned Aerial Vehicle

ACRONYM	DESCRIPTION
VA	Virginia
WOz	Wizard of Oz